AL-TP-1992-0012

ARTIFICIAL NEURAL SYSTEMS APPLICATION TO THE SIMULATION OF AIR COMBAT DECISION MAKING

Jeffrey J. Roorda
Michael X. Crowe

Ball Systems Engineering Division
5580 Morehouse Drive
San Diego, CA 92121-1709

HUMAN RESOURCES DIRECTORATE
AIRCREW TRAINING RESEARCH DIVISION
Williams Air Force Base, AZ 85240-6457

April 1992

Final Technical Paper for Period September 1988 – November 1991

Approved for public release; distribution is unlimited.

ARMSTRONG LABORATORY

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5000

NOTICES

This technical paper is published as received and has not been edited by the technical editing staff of the Armstrong Laboratory.

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Office of Public Affairs reviewed this paper, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This paper has been reviewed and is approved for publication.

RICHARD A. THURMAN
Contract Monitor

DEE H. ANDREWS, Technical Director
Aircrew Training Research Division

LYNN A. CARROLL, Colonel, USAF
Chief, Aircrew Training Research Division

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1992	Final – September 1988 – November 1991

4. TITLE AND SUBTITLE

Artificial Neural Systems Application to the Simulation of Air Combat Decision Making

5. FUNDING NUMBERS

C – F33615-88-C-0006
PE – 62205F
PR – 1123
TA – 35
WU – 12

6. AUTHOR(S)

Jeffrey J. Roorda
Michael X. Crowe

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ball Systems Engineering Division
5580 Morehouse Drive
San Diego, CA 92121-1709

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

Armstrong Laboratory
Human Resources Directorate
Aircrew Training Research Division
Williams Air Force Base, AZ 85240-6457

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AL-TP-1992-0012

11. SUPPLEMENTARY NOTES

Armstrong Laboratory Technical Monitor: Dr. Richard A. Thurman, (602) 474-6561

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The research goals of this project were to ascertain the applicability of Artificial Neural Systems (ANS) technology to expert systems tasks in general and to support the simulation of Air Combat Maneuvering (ACM) decision-making in the training environment. In the experiments conducted under this program, neural networks have aptly displayed their unique capabilities to overcome some of the more difficult aspects of knowledge engineering. ANS approaches have been shown to be capable of producing robust, generalized solutions even under novel circumstances. By capturing and simulating the expertise of human pilots in a neural network, students may be provided with expert training devices which may come very close to the look and feel of real air-to-air combat. It is expected that ANS technology will continue to provide new solutions to the simulation of human performance for training purposes.

14. SUBJECT TERMS

Air combat maneuvering
Artificial intelligence
Artificial neural systems
Decision making
Flight simulation
Flight simulators
Neural networks

15. NUMBER OF PAGES

100

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

CONTENTS

CONTENTS (CONTINUED)

LIST OF FIGURES

LIST OF FIGURES (CONTINUED)

LIST OF FIGURES (CONTINUED)

LIST OF FIGURES (CONTINUED)

LIST OF TABLES

PREFACE

This final technical paper for the Air Combat Maneuvering Expert System (ACMES) Program Research and Development Announcement (PRDA) was prepared for the Armstrong Laboratory, Human Resources Directorate, Aircrew Training Research Division (AL/HRA) under Contract Number F33615-88-C-0006 with Ball Systems Engineering Division (BSED). The research reported herein was performed between 29 September 1988 and 30 November 1991. The government program monitor for this task was Dr. Richard A. Thurman (AL/HRA). Additional programmatic support was provided by Dr. Byron J. Pierce, Dr. Wayne L. Waag, and Dr. Thomas H. Killion, all of AL/HRA, and by Mr. Bart Raspotnik of Logicon, Inc.

In addition to this paper, documentation of the ACMES includes an executive summary, a user's manual, an analyst's manual and a programmer's manual.

ARTIFICIAL NEURAL SYSTEMS APPLICATION TO THE SIMULATION OF AIR COMBAT DECISION MAKING

PROGRAM OVERVIEW

Program Objectives

The stated objective of Program Research and Development Announcement (PRDA) 87-7, entitled "Expert Systems Approach to Modeling Pilot Decision making in Air Combat Maneuvering" is to design, develop, and validate a computer-based expert model of pilot decision making in air combat maneuvering (ACM). PRDA 87-7 is part of a program which is geared toward the eventual production of an expert system trainer capable of providing ACM decision training to F-15, F-16 and T-38 pilots. An expert model of air combat maneuvering would enhance the effectiveness and consistency of ACM decision-making training. The concept is to provide the student and instructor pilots (IPs) with an ACM training station which displays the interactions of the student's aircraft with a simulated adversary aircraft. The ACM expert model would control the adversary aircraft's reactive maneuvers, resulting in behavior that is similar to an experienced pilot in the identical tactical situation. In addition, such a system could also be used to demonstrate correct maneuvering of the student's aircraft. For example, when observing the student perform a maneuvering error, the IP could stop the simulation, back it up a few seconds, and let the expert system take over to show the student pilot a better sequence of maneuvers.

Armstrong Laboratory's Aircrew Training Research Division (AL/HRA) contracted Ball Systems Engineering Division (BSED) to focus its efforts under this PRDA on the development of the ACM Expert System. Many techniques have been explored in the past for capturing, representing, and recalling knowledge in a fast and reliable fashion, but all have suffered from major inadequacies in terms of knowledge acquisition, speed of recall, generalization, knowledge base size, and the ability to handle incomplete or inaccurate data. Though some improvements have been made using traditional artificial intelligence (AI) methods, current systems continue to become overwhelmed by size and speed constraints, and many of the aforementioned technical hurdles remain. The field of Artificial Neural Systems (ANS), also known as Connectionism and

Artificial Neural Networks, has shown much promise in overcoming some of the more intractable elements of simulating intelligent behavior. Therefore, BSED chose Artificial Neural Systems as the preferred approach for accomplishing the objectives of the ACM Expert System PRDA.

Overview of ACM Decision Making

The intent of the ACM Expert System is to simulate the decision behavior of fighter pilots who have acquired a certain proficiency in air-to-air combat. By directly modeling human experts, the details and nuances of successful ACM performance can be used to train others to achieve that level of performance. In an air-to-air engagement, once an adversary aircraft has been detected, the pilot begins a decision process based both on his own goals and expectations, and on the actions of the adversary. The basic goal of an aircraft pilot during air combat maneuvering is to destroy the adversary aircraft while simultaneously avoiding the destruction of his own aircraft. They accomplish this goal by maneuvering their aircraft through three-dimensional space in an effort to obtain a position (relative to the adversary aircraft) which allows the adversary aircraft to be attacked. At the same time, they must avoid the adversary's attempts to maneuver into an attack position against their own aircraft.

Though the above description provides an adequate top-level outline of the fundamentals of air combat, there are a number of other factors which influence the performance of ACM, such as the limitations of the aircraft weapon systems, the use of countermeasures, weather conditions, and the prevailing rules of engagement. A fielded ACM training system would require that these variables be taken into account to provide the proper "success" criteria profile which would emphasize and build upon all the relevant elements of effective maneuvering and weapons employment. However, the basic requirement of air combat is the generation of a successful sequence of maneuvers based on the relative geometry of the engaged aircraft. A "successful" engagement is one in which the elimination of the threat is coupled with the survival of the ownship. This is the framework upon which the ACM Expert System was established.

Scope of this Effort

BSED's effort under this PRDA has been focused to specific neural network approaches within a specific ACM domain. The objective for the ACM Expert System is the production of realistic air combat maneuvers under within visual range (WVR) tactical situations. Beyond-visual-range (BVR) maneuvering, countermeasures utilization, and weapons employment are outside the immediate scope of this effort. Furthermore, though it is feasible to apply the approaches used in this effort to the multi-ship arena, the ACM expert system was designed for one-versus-one engagements.

In terms of neural network research, no attempt was made to provide a comprehensive assessment of various network paradigms. Rather, a single paradigm (with some variations) was used for all neural network experiments. The intent here was to focus on the ANS approach with the highest potential for building a working solution to the ACM Expert System, and to demonstrate in detail the usefulness and limitations of neural network technology for the simulation of ACM decision making.

There are two basic issues at the design level which have been used to guide BSED's development of the ACM Expert System: (a) What does the pilot need to know when making ACM decisions and (b) What is the form of the pilot's decision output that leads to changes in the aircraft's flight path? At the implementation level, the overriding question is: How can neural network technology best be applied to bring about new solutions in the simulation of ACM decision making? The goal of this effort was to create a system which takes situational data as input and combines it in the proper way to produce a reasonable and realistic maneuver as output. Input data takes the form of relative geometry and specific aircraft parameters which the pilot might use and would have available during a real ACM engagement. The chosen form of output control is the amount of heading, pitch, and velocity control required throughout the flight envelope. The resulting system provides a working framework for the evaluation of neural networks which simulate air combat maneuvering and allows for the extension of these results to other simulations of expertise and training environments.

THE NEURAL NETWORK APPROACH

Introduction to Artificial Neural Systems

BSED's technical approach to simulating pilot decision-making expertise involves the use of Artificial Neural Systems (ANS) technology, a relatively new approach for information processing. ANS, commonly referred to as **neural networks**, provides a methodology for combining or associating data or knowledge through a "self-organization" of the representational system. In other words, a neural network produces a mapping which mathematically relates the input space to the output space. This mapping technique may be applied to any problem where the underlying function of association is complex or unknown. As applied to the ACM Expert System, this mapping capability is utilized to model human performance by learning the association between the tactical situation and the correct maneuver response. Rather than capturing expertise as a set of logical "if-then" rules, as is done in traditional artificial intelligence (AI) expert systems, a neural network develops expertise by adapting its internal arrangement in response to examples of expert behavior. While some existing ACM simulations rely on the diagnostic, pattern recognition capabilities of AI expert systems, others use a value-driven, trajectory prediction technique to determine a course of action. An ANS-based system can combine these approaches in that the system can learn to associate the recognition of a pattern with the selection of the proper course of action.

In BSED's ACM Expert System, the neural network creates a mapping or association from the tactical situation (input space) to the appropriate maneuver response (output space), as is shown in Figure 1. To serve as a source of expertise for the ACM Expert System, engagement profile data was selected from the Simulator for Air-to-Air Combat (SAAC), which is a man-in-the-loop, multidome simulator at Luke AFB, Arizona. The underlying assumption of this expert system development process is that given a sufficient representation of the input parameters, a general relationship exists between tactical conditions and maneuver responses that can be resolved and duplicated computationally. By using the performance of pilots during SAAC engagements, it was anticipated that representative examples of this input/output relationship could be collected and used to train a neural network.

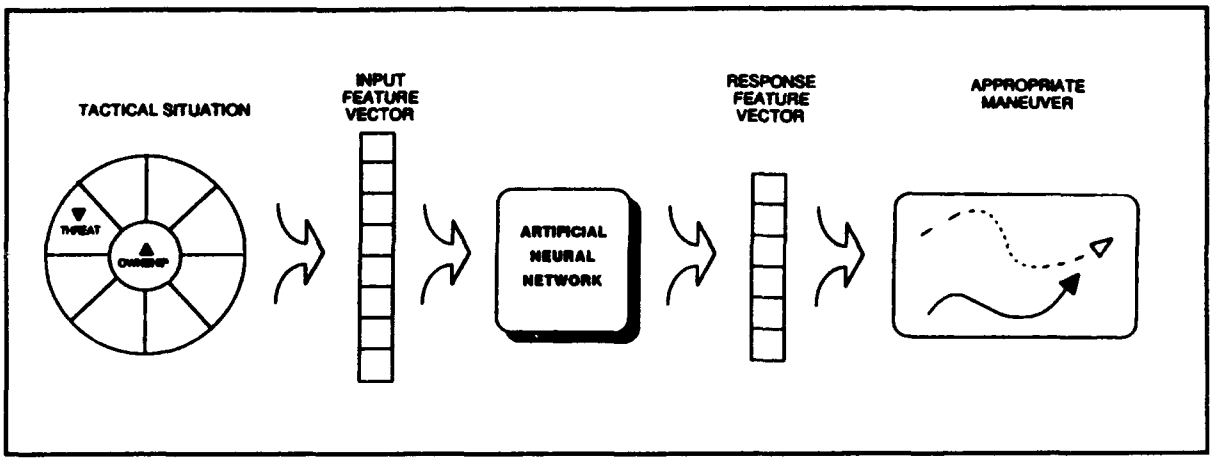


Figure 1. BSED's approach to the ACM Expert System using a Neural Network.

The internal arrangement and behavior of a neural network which allows such a mapping to be accomplished is drawn from the massively parallel and highly distributed processing arrangement found in the brain. Specifically, neural networks are biologically motivated models of information processing. They use the interactions of simulated neurons to store, recognize, and recall knowledge. Neural networks are often constructed as a hierarchy of layers. Each layer contains some number of simulated neurons, technically known as **processing elements (PEs)**, which are interconnected throughout the network. The strengths and structure of these interconnections are what determine the system's ultimate operation. The strength of a connection, the extent to which one PE affects another, is known as the **weight** of the connection. Figure 2 is a pictorial representation of a typical processing element in a neural network.

In Figure 2, processing element j receives some number of inputs, i, from other PEs in the network. The levels of each of these input signals, x_i, are multiplied by their connection weights, w_{ji}, and summed together to produce S_j, the total signal into j. In general, this summation represents the external influence of other PEs on PE j. S_j is used to calculate an update value to $A_j$new, the current level of activation, or state, of PE j. The new activation level is then applied to a threshold function to determine O_j, the output signal to be produced by j. The output is then received by other PEs in the

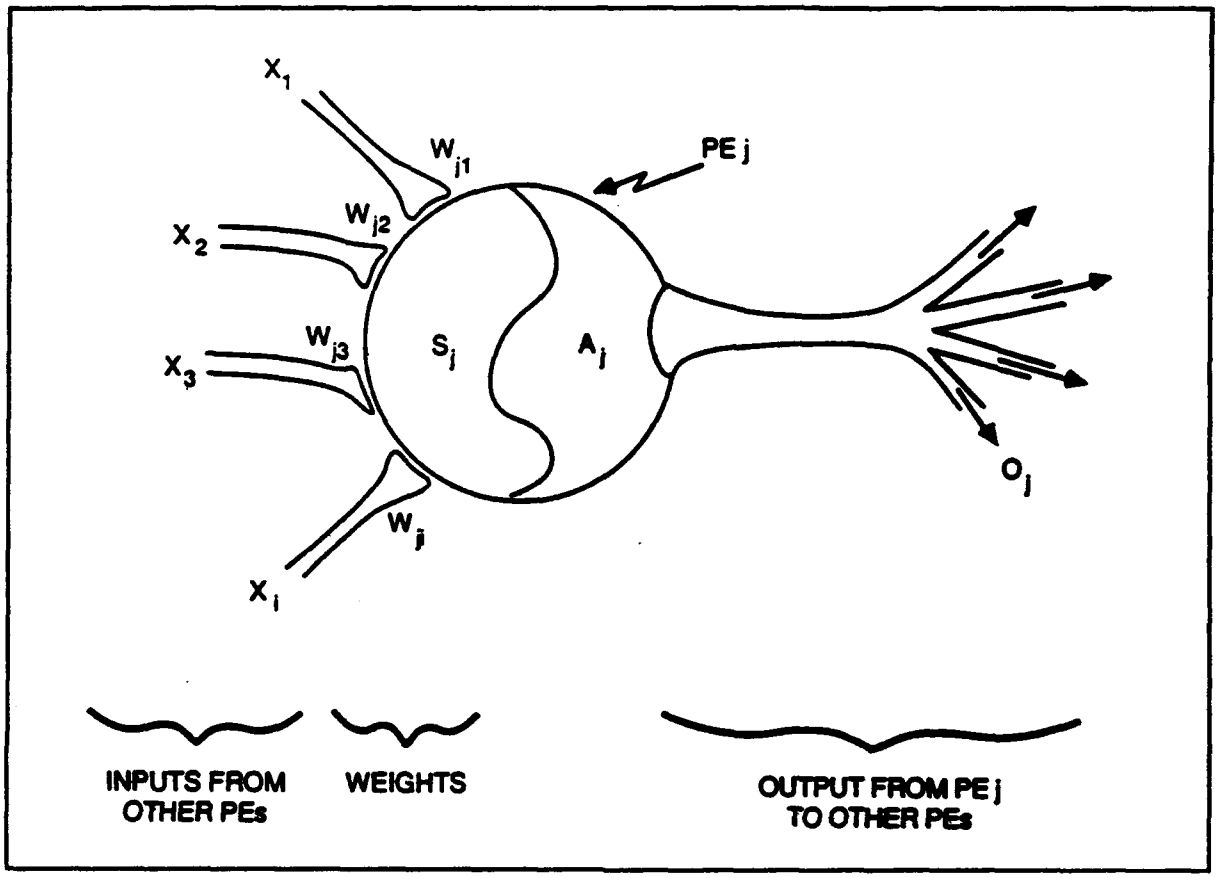


Figure 2. Schematic Representation of a Typical Neural Network Processing Element.

network. The activity of an individual PE as described above can be represented mathematically by the following equations:

$$S_j = \sum_i x_i w_{ji}, \qquad (1)$$

$$A_j new = f(A_j old, S_j), \qquad (2)$$

$$O_j = g(A_j new) \qquad (3)$$

where S_j is the summation value of PE j, $A_j new$ is the new activation level of PE j, f() is some activation function which produces the new activation based on the current activation and summed input, O_j is the output of PE j, and g() is some thresholding function for determining the level of output. Thresholding equations generally have a sigmoidal form as in equation (4).

$$O_j = \frac{1}{1 + e^{-A_{jnew}}} \qquad (4)$$

Since the weights and the PE activation levels can have both positive and negative values, the processing elements can have both excitatory and inhibitory influences on other PEs in the network. PE output levels are usually continuous values scaled between 0 and 1 or between -1 and +1. Weights vary over a wider range of values. A large absolute weight value represents more potential influence among PEs. A weight value of zero indicates that no connection exists between those two PEs.

This approach to information processing differs significantly from traditional computing methodologies. For example, most computers are based on the operation of a single, complex central processing unit, the CPU, whereas neural networks utilize the effects of many, simple processing elements. Traditional computing is done in a step-by-step, serial fashion, while neural networks operate through a parallel update of the PEs. On traditional computers, solutions are programmed via algorithms which are executed as a series of instructions. The neural network has no algorithm; it learns how to operate through examples of correct behavior and the resulting knowledge is stored in the structure of the network. Finally, traditional computing approaches provide precise answers for problems where the underlying function or algorithm is known to the programmers. Neural networks usually provide a more general solution to problems where the underlying algorithm is not known or is extremely complex.

The Multilayer Back-Propagation Paradigm

Many ANS paradigms have been developed to accomplish the determination of a mapping between input and output data; some are better understood than others. The specific neural network paradigm used for the ACM Expert System representation is the **Multilayer Back-Propagation Network (MBPN)** which is the most common and most successful neural network paradigm in current usage. The internal rules and procedures for how the MBPN network arrives at a set of associations for ACM performance need not be specified by the programmers. Rather, ACM expertise is represented in the patterns of activations and the weighted connections of the network's processing elements. The neural network

requires only that the problem be represented in terms of an input vector which represents the current tactical situation, and a corresponding output vector which determines the correct maneuver response. To achieve the proper set of weights and activations, a back-propagation network is **trained** by being exposed to examples of correct performance. During the network's learning process, expert performance is used to generate the values of the input and output vectors which are then **clamped** to certain processing elements of the neural network as their activation levels. Through a series of associated input and output examples, these clamped activation levels methodically influence and adjust the rest of the network until a general solution to the mapping between input and output is found.

In the back-propagation paradigm, there are three or more layers of processing elements. The first layer is the input layer to which (in the case of the ACM Expert System) the tactical situation is presented, and the last layer is the output layer which produces the maneuver response. The component values of the input and output vectors are represented by the activation levels of the PEs in those layers. Between the input and output layers are one or more **hidden** layers of PEs which are responsible for building up explicit and implicit associations between the input and output feature vectors. This multilayer arrangement allows associations of a more abstract nature to be formed than would be possible with just two connected layers.

Figure 3 is a diagram of a typical three-layer back-propagation network. The weights in such a structure would occur where connections are made from the input layer to the hidden layer and from the hidden layer to the output layer. It should be noted that most back-propagation implementations include a **bias element** which has a constant activation level of 1.0 and makes a single weighted connection to each PE of the hidden and output layers. The bias weights are included in the structure to allow the network to reproduce a broader class of mappings.

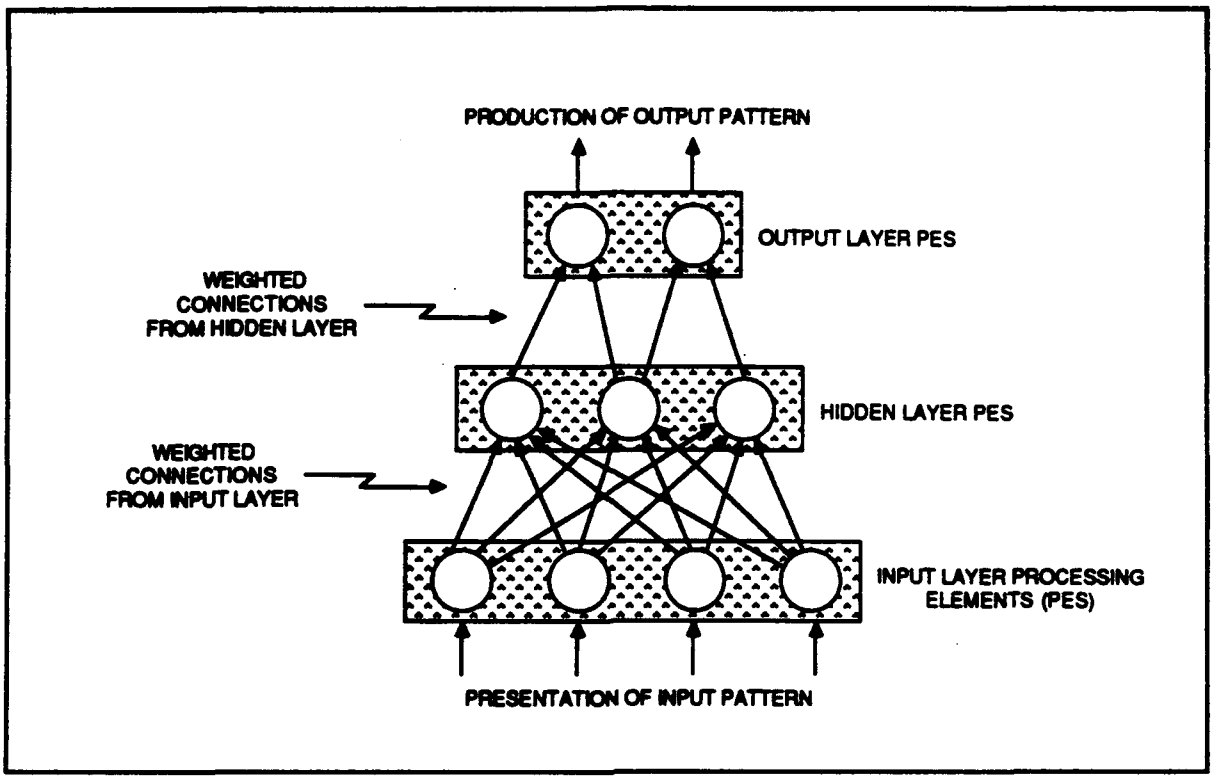


Figure 3. Structure of a Typical Three-Layer Back-Propagation Network.

Back-propagation works by employing a concept known as the **generalized delta learning rule.** This means that the neural network is trained to associate specific sets of input and output data by being taught the difference, or **delta**, between the results it produces and the actual desired result. Neural network training simply means repeatedly presenting the network with correct examples of associated input and output data and allowing the system to adjust itself when mapping errors occur. Naturally, the errors will be quite large when training begins and will gradually decrease as training continues. When fully trained, the network will be capable of reproducing the response performance of the data in the training set. In many cases, a properly trained network is capable of providing the correct output response to "noisy" input data or even to input data upon which it has never been specifically trained.

Problem Representation Using Neural Networks

One of the key elements of any neural network implementation is the representation of certain key parameters at the input and output layers. Next

to the selection of training data, the input/output representation selected for the neural network embodies the majority of knowledge engineering required to produce a successful system. The following section describes the process of constructing a problem representation with an MBPN neural network.

Representation Factors

A critical element in the design and development of a neural network is the selection of specific components of the problem domain to be represented in the form of the input and output vectors. It is important that the problem be represented by input and output components which are relevant to the real-world association between those elements and for which an underlying mapping exists. Furthermore, an initial representation system must be chosen carefully to ensure that this initial system will eventually scale up successfully to represent all facets of the full problem. If the final representation must be altered drastically from the prototype, the validity of the initial representation may not carry through to the later forms. As an example of how to build a neural network representation, consider the ACM Expert System domain. At the input layer, an initial representation could be approached by determining a minimal set of tactical input data that might be required by a combat pilot when making ACM decisions in a one-versus-one engagement. Similarly, the components of the output response might be chosen to meet a minimal set of aircraft flight control commands to direct aircraft maneuvering. In addition to these domain-specific requirements for sufficient representational accuracy in a neural network, there are some other important driving forces behind the selection of specific representations for the input and output vectors.

First, it is desirable to maintain a reasonable working size for the initial network. There is a tendency to place all possible decision criteria in the input layer and let the network decide what is important and what is not. The initial representation should be limited to only the most crucial elements of the input/output mapping. Having established this initial limitation, one must keep in mind that the chosen representation scheme must be capable of being scaled up to model additional input and output parameters if necessary. Also, the input/output design should ensure that the representation in the network is compatible with data available for network

training. In the case of the ACM Expert System, this means that the network representation should be compatible with or a subset of available SAAC data upon which system training is to be based. At the implementation level, the network designer must decide whether to use the values of the input/output components directly in the network or utilize some transformed representation.

Structure of the Input and Output Layers

For the ACM Expert System, the structure of the input vector should be based primarily on what the pilot (and presumably the expert system) needs to know about the prevailing tactical situation to arrive at a reasonable maneuver response. More specifically, the input vector should represent the top-level, dynamic variables which define the tactical situation between the competing goals of the pilot and adversary aircraft. The term **top-level** refers to situational data that is directly perceived by the pilot from the environment which relates to the present situation. Relative positions and orientation of the two aircraft are examples of top-level data. An example of data that is *not* top-level would be whether the current geometry is defensive, offensive, or neutral for the pilot. This implicit **meta-level** knowledge will be discussed later when the internal representations of the model are described. **Dynamic** data are those parameters which change in value over the course of the ACM profile. Non-dynamic data may be excluded from the input vector because the system will simply learn to operate within the constant constraints dictated by these factors where they are present in the training data. For example, the aerodynamic limitations of the aircraft need not be spelled out specifically to the neural network via the input vector, yet will become an implicit part of the simulation's maneuver responses by virtue of their existence in the responses found in the training data. Of course, like the human pilot, such a neural network would likely have to be retrained to effectively fly a different aircraft with different aerodynamic limitations.

The output of the ACM neural network is the output or response vector which represents how the simulated pilot should maneuver the aircraft under the prevailing tactical conditions. Not only must the output vector provide a sufficient representation of the maneuver response, it must also be capable of being translated into a set of parameters that can drive the operation of

an aerodynamic model. This is how the ACM neural network influences events in the operating environment.

Neural Network Training

Neural networks are trained to accomplish a desired mapping between input and output states by being repeatedly exposed to examples of the input/output association. This training process is dependent on the correct structuring of the hidden layers of the network, the selection of various learning parameters of the back-propagation algorithm, the selection of appropriate examples to make up the training set and the optimization of the duration of exposure to the training examples.

Structure of the Hidden Layers

The ability of a neural network to successfully learn and recall the proper mapping upon which it has been trained is based entirely on the structure and connection weights of the system. The input and output layers provide the interface to the outside world, and through them, the network is trained to accomplish the desired mapping. However, it is the internal structure, the connections to, from, and between the hidden layers of the neural network where the system learns how to produce the proper responses to certain input conditions. In the ACM Expert System, these internal connections and weights embody the expertise of the simulated pilot.

While the representations of the input and output vectors are made very explicit prior to training the network, it is difficult to predict with certainty what the processing elements and interconnections of the middle layers will come to represent. After training the ACM network, however, there will be certain **emergent** properties of the network's internal conformation which will represent the abstract nuances of successful ACM performance. Meta-level knowledge structures like maintaining speed, building energy, and gaining lateral separation, become embodied in the activation levels and connection weights of the system without being explicitly programmed by the system designers.

This is one of the fundamental benefits of the neural network approach. Determining how a human expert combines the information received by the brain into a useful response is very difficult to spell out as a set of rules. The amount of knowledge, and the types of combinations of that knowledge, are vast. Often, the expert is not capable of furnishing a detailed explanation of precisely why a certain action was taken. But the neural network, with the proper representation and training, will self-organize to arrive at its own understanding of how decision responses are formed. The weights and connections of the hidden layers will eventually store the decision strategies, which are learned automatically through example, as an implicit set of internal rules.

Weight Adjustment Process

Supervised neural network training, as is used in the back-propagation algorithm, consists of repeatedly providing the system with input data, observing the response at the output layer, and using the difference between the current and desired output values to alter future responses in the desired direction. This is accomplished by propagating output errors back through the layers of PEs and making the appropriate weight adjustments where necessary. Each weight adjustment moves the overall state of the network in the direction of a global solution to the input/output mapping. Such a procedure obviously requires a known set of correlated input/output vectors which can be used to train the network.

During the neural network training process, input feature vectors which represent the current decision conditions are presented to the network. From this data, the network uses equations (1), (2), and (3), previously listed, to arrive at an appropriate output response. This calculated response is then compared to the correct, or target response from the training data, and the difference between the two, the error, is fed back into the network to correct the association between input and output. In effect, this teaches the network which output to produce when it sees that kind of input data again. As the process continues in this fashion, the neural network will eventually provide the correct response for each input situation upon which it was trained. At this point, training is considered complete for this network, and it may be used in a feed-forward mode where no back-propagation of errors is required.

Back-propagation training is a multi-step process, each step providing adjustments to a single layer of interconnections. Assuming a three-layer system as shown in Figure 3, the first back-propagation of errors, or deltas, produced at the output layer of PEs, adjusts the weights of the connections between the hidden layer and the output layer. The weight change equation governing this adjustment has the form

$$\Delta w_{ji}(t+1) = \alpha(\delta_j s_i) + \beta\Delta w_{ji}(t) \qquad (5)$$

where $\Delta w_{ji}(t+1)$ is the current weight change to be made to the connection from the ith hidden element to the jth output element, $\Delta w_{ji}(t)$ is the previous weight change made to this same connection, β is a momentum term which determines the effect of past weight changes to the current adjustment, α is a learning rate term which determines how large each adjustment should be, and s_i is the activation value of the ith hidden element projecting to the jth output element. When calculating the deltas for the output elements, the δ_j term in the above weight change equation is defined as

$$\delta_j = (t_j - o_j)f'_j(net_j) \qquad (6)$$

where t_j is the target activation value for the jth element of the output pattern, o_j is the actual activation value produced by the system at the jth element of the output pattern, and $f'_j()$ is the derivative of a "squashing" function which operates on net_j, the sum of the inputs into output element j. In most cases, the squashing function is a sigmoidal function of the form

$$o_j = f_j(net_j) = \frac{1}{1 + e^{-net_j}} \qquad (7)$$

and the derivative of this function would then be

$$f'_j(net_j) = o_j(1 - o_j) \qquad (8)$$

where $net_j = \sum_i s_i w_{ji}$. The sigmoidal form used in (7) is also used as the thresholding function for calculating activation and output from the sum of inputs as given in equations (2) and (3). Figure 4 shows how this delta calculation and weight adjustment is carried out for the hidden-to-output connections. These updates are also carried out for the bias weights, though the single processing element in the bias slab maintains a constant activation value of 1.0.

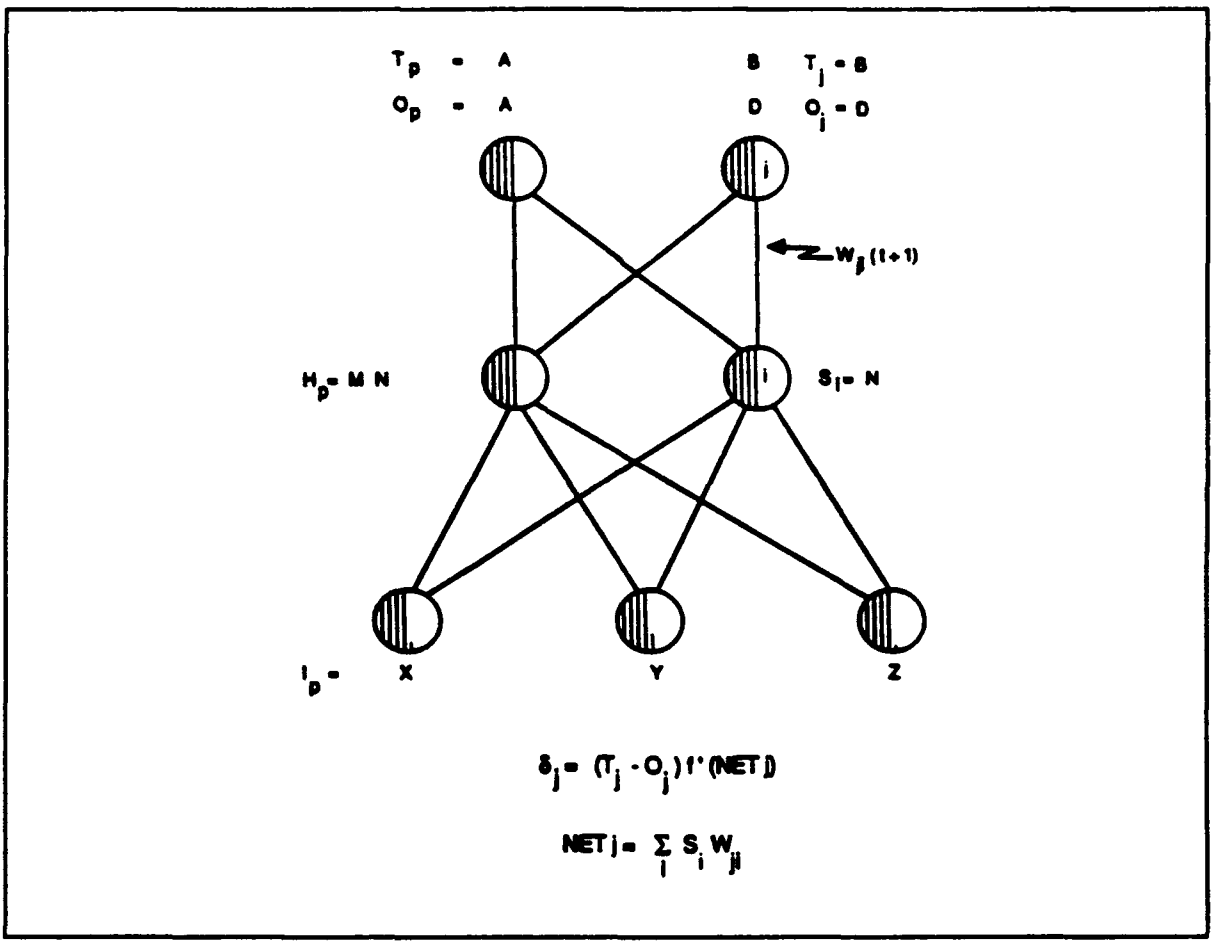


Figure 4. Provided with input pattern (X Y Z), the network has produced output pattern (A D) via the hidden pattern (M N). The target output pattern for this input is actually (A B), so the delta between t_j and o_j is used to adjust the weight of the connection between the hidden layer element i and the output element j.

In the second step of the three-layer back-propagation training process, weight adjustments are made to the connections between the hidden layer of PEs and the elements of the input layer. The weight adjustment process uses the same formula as defined in equation (5), but at this stage, the term $\Delta w_{ji}(t+1)$ refers to the current weight change to be made to the connection from the ith input element to the jth hidden element, and $\Delta w_{ji}(t)$ is the previous weight change made to the same connection. The ß and α terms are the same, but s_i now refers to the activation value of the ith input element sending output to the jth hidden element. When calculating the deltas for the hidden elements, the δ_j term in equation (5) is defined as

$$\delta_j = f'_j(net_j)\sum_k \delta_k w_{kj} \qquad (9)$$

where $f'_j()$ is the derivative of the "squashing" function which operates on net_j, the sum of the inputs into hidden element j from the input elements, and $\Sigma\delta_k w_{kj}$ is the sum of the previously computed deltas of the k output elements which connect to hidden element j multiplied by their connection weights, w_{kj}. Figure 5 shows the input-to-hidden layer delta calculation and weight adjustment process.

The back-propagation algorithm is a **gradient-descent** heuristic, which means that the weight changes will minimize the squares of the differences (error) between the actual and the target output values. This error function is known as **Mean Squared Error (MSE)**. The back-propagation process attempts to move across the multidimensional weight space so as to continually reduce this error function. The weight space may be visualized as a landscape with various wells, valleys, hills, and ridges. Somewhere in this landscape is a lowest point or **global minimum**, which represents the optimal performance of the network. The MBPN process adjusts the weights so that the weight surface is traversed in the steepest fashion. However, it does not guarantee that the global minimum will be found; the process may get trapped in a valley which represents only a **local minimum**. Finding the global minimum is the goal of network processing, and the techniques for doing so (and avoiding local minima) will be discussed later. One cycle of the network's operation, including weight updates if necessary, is referred to as a single iteration. The entire training process may take thousands or even millions of such

iterations before the system settles into a minimum well, and the proper mapping is learned.

Since the gradient descent process attempts to minimize Mean Squared Error, the most common method for measuring the performance of a back-propagation network during training is to calculate the Mean Squared Error over the entire set of training data. The value for MSE is calculated using the following equation:

$$MSE = 1/N \sum_{k}^{N} \sum_{j}^{M} (t_j^k - o_{Lj}^k)^2 \qquad (10)$$

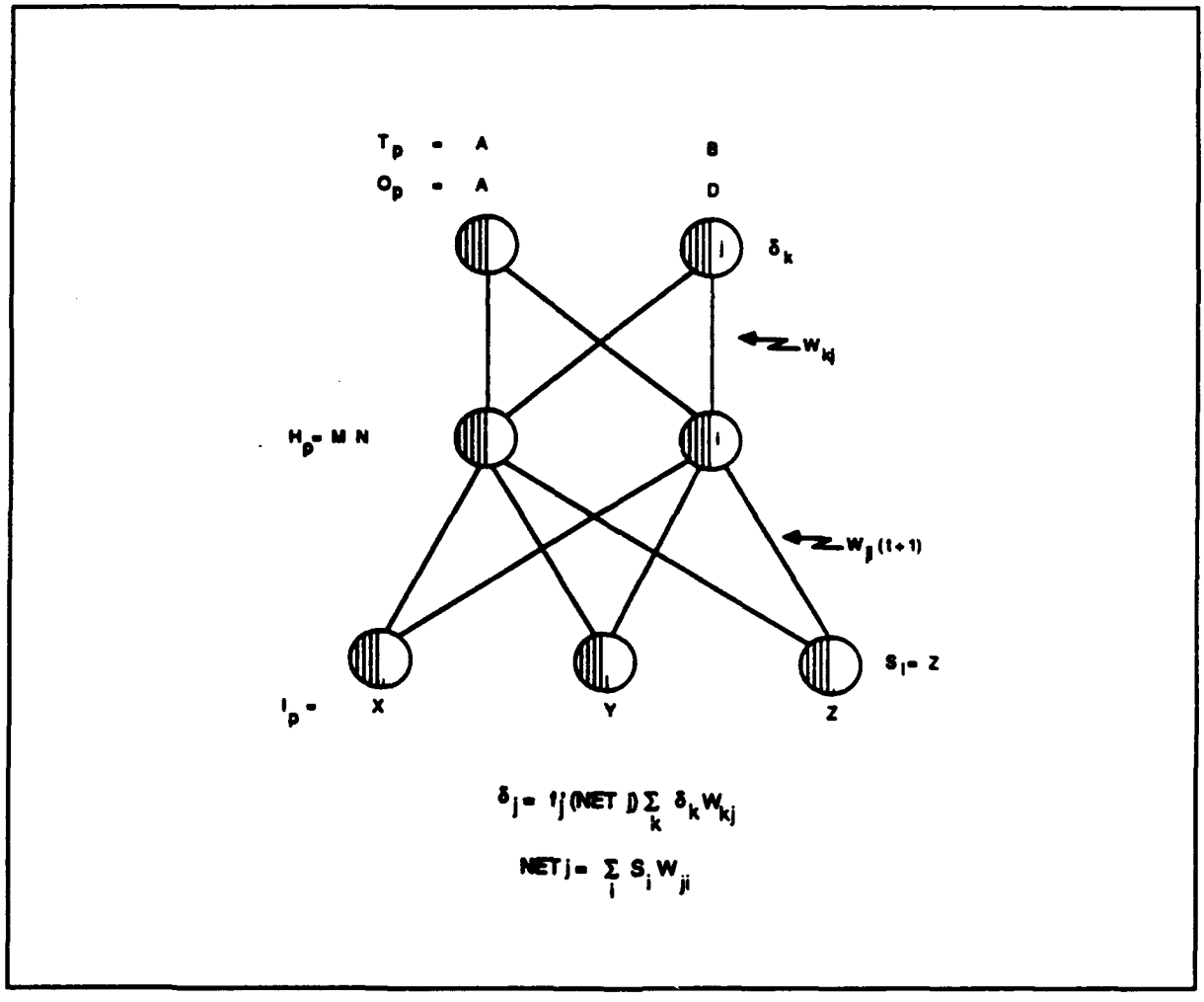


Figure 5. During the second step of the back-propagation process, the sum of the output element deltas multiplied by their connection weights to hidden element j is used to adjust the weight of the connection between the input layer element i and element j of the hidden layer.

where L is the index of the output layer, M is the number of elements on the output layer indexed by j, k is the index of iterations, and N is the number of consecutive iterations since the error calculations were last initialized. MSE is only one Measure of Effectiveness (MOE) for the performance of the ACM neural network, and other validation techniques will be discussed in the section titled 'Artificial Neural System for the Representation and Collection of ACM Decision-Making Expertise.'

Batching is one additional variation sometimes used in MBPN training. When using batching during the training process, a selected number of delta weight calculations are accumulated and averaged prior to weight adjustment. For example, with a batch size of 100, the system is exposed to 100 training associations before the cumulative effect of the delta weights makes its weight adjustment. The intent of the batching process is to accumulate a more representative gradient direction from a number of samples rather than basing the weight change decisions on each individual training association. However, since the overall number of weight updates is reduced when batch sizes are greater than one, it may take longer for the network to converge to a desirable level of response.

Selection of Training Data

The ultimate effectiveness of a neural network to reproduce the decisions the expert is able to make is fundamentally dependent on the set of data used to train the network. Since the desired outcome is the simulation of pilot performance, it is necessary to locate a source of consistent and controllable performance data. Before obtaining training data, certain preselection criteria must be met. In order to facilitate the learning of the network, the training data should be selected to be as consistent as possible and decision conditions present within the data should not take place under conditions not represented by the input vector. For example, the network should not be trained to maneuver with and without weapons if the input representation provides no indication to the system as to weapons availability. It is also useful to select data at a consistent level of proficiency, presumably from consistently "good" performances, to the extent that this is possible.

Another consideration is that the selected training data adequately span the entire solution space. Exactly what the solution space is can be difficult to specify in complex, multidimensional problems like the ACM representation. In general, the solution space consists of all the possible associations of input with output. The important point regarding the solution space is that training data is selected to represent a relatively uniform distribution over the entire space. To better understand the concept of a solution space, examine the relatively simple ANS experiments described in the paragraph on 'Preliminary Neural Network Experiments and Results.'

Unless the solution space can be specified at some level of detail, one is forced to choose data in a random fashion from real-world examples and assume that this selection will adequately span the possible conditions to which the network will be required to respond. When the training data is selected appropriately, the network will form a mapping which allows it to generalize solutions from the relatively small set of training examples. This means that the neural network will be capable of responding correctly to novel situations that were not seemingly reflected in the training data. The ability of neural networks to generalize successfully from the training data is clearly illustrated in the experiment described in the Angle Proximity Calculator paragraph, and is discussed in some detail with regard to the Lead Pursuit/Intercept Simulation system.

Optimization of Training Parameters

In addition to the creation of an optimal network training file, one must determine the most effective structure for the hidden layer(s) and to fine-tune the learning equations. There are certain useful heuristics in the development of the neural network's internal structure, but discovering and applying those heuristics is an acquired skill in these early days of ANS research. The number of PEs used in the hidden layer(s) is partially a speed-versus-accuracy question in that more PEs means more connections and, therefore, longer training cycles. At the same time, a sufficient number of PEs must be present to successfully accomplish the desired mapping. Depending on the specific association that is being learned, the number of PEs and connections directly determine the network's capacity.

The exact capacity of the network, or number of associations learned, is difficult to measure since the network is expected to generalize over many more associations than it is actually trained. The question of capacity is more correctly stated as the performance of a network on a general set of test data. This performance may be measured by the minimization of Mean Squared Error (MSE), or by some other domain-relevant scoring metric. In general, the number of hidden PEs is gradually increased from one until the optimal performance is found. Another factor, however, must be taken into consideration. If too many internal PEs are present, the system may "memorize" the training data associations, but be completely unable to generalize to the broader scope of the problem. When this happens, the network will exhibit very good performance on the training set, but a broader test set will show poor correlation between input and output.

The rationale for multiple hidden layers in neural networks is that higher, abstract concepts may be encoded. The idea is that multiple layers allow for additional ways for the network to divide up the problem space. BSED has been unable to ascertain in any detailed, consistent fashion that the performance of a four-layer network is better than a three-layer system for the same problem.

After both the training file and the network structure have been defined, the next step is to find the optimal values for each parameter in the network initialization and training process. For example, when the network is created, the weights must be assigned some initial values before training begins. Without any previous data to guide this initialization, the best approach is to simply assign random values to the weights. However, both the seed value for the random number generator and the allowable range for the random values themselves may affect the eventual performance of the trained network. In effect, they determine the starting point for the gradient descent process on the weight surface. Due to the complexity of the weight surface in most large networks, there are probably many paths to the global minimum. Consequently, the value chosen for the random seed has little effect on the final performance of the network, as long as training time is long enough. The optimal range for the initial weight values can have a greater effect, but may also be determined in a more systematic fashion.

When choosing a range for the initial weights, a primary heuristic is that small initial weight values (less than ±1.0) create a network that is generally less "committed" to any specific set of input values. In a sense, this makes the network more truly random. Initial ranges are usually set between ±1.0 as a default value. By trying a few different range values around ±1.0 and observing the pattern of Mean Squared Error reduction after a fixed number of iterations, a curve with a minimum point of inflection can be plotted. The minimum point indicates the lowest MSE, and thus provides an indicator for the optimal range for the random initial weight values. A similar process may be followed for the learning rate, α, the momentum term, ß, the batching size, and the steepness of the sigmoidal thresholding equations. Each value is chosen so as to optimize (in a local fashion) the reduction of MSE for a given training set.

It should be noted that this optimization of training parameters based on minimization of MSE over a fixed sample set provides only a rough indication of the "best" values. However, experience has shown that approximate values for these parameters are all that is required for successful learning capability in the network, if learning is possible at all. In other words, small variations in these parameters beyond the rough estimate usually provides very little additional improvement in MSE reduction. The full set of parameters that may be optimized for MBPN training is shown in Table 1.

Table 1. Optimization Parameters for Neural Network Training

Alpha	- Learning Rate
Beta	- Smoothing or Momentum Term
Batch Size	
Network Structure	- Number of layers, Number of PEs and Connections From Inputs to Outputs
Initial Weight Range	
Random Seed for Initial Weights	
Activation Function Type	
Steepness of Activation Function	

The optimal duration of training is best determined by observing MSE as training progresses and by noting the absolute differences in the desired and computed output values. In general, it is desirable to continue training until the output deltas in the training set become very small. Every problem will require a different amount of training to achieve the desired level of performance. Where a clear correlation exists, and if the data set has been selected properly, the network will perform most effectively when it has minimized MSE for the training set. This is apparent when MSE is very small and no longer changing. The gradient descent algorithm proceeds asymptotically, so the vast majority of learning occurs at the beginning of training. This phenomenon is shown pictorially in Figure 6. It is possible, however, to "overtrain" the network, causing it to lock in on the training samples and lose its ability to generalize. This effect is similar to the case when too many hidden units are used. Also, with complex networks, MSE may not decrease monotonically, and MSE alone may not provide a sufficient guide for network performance measurement.

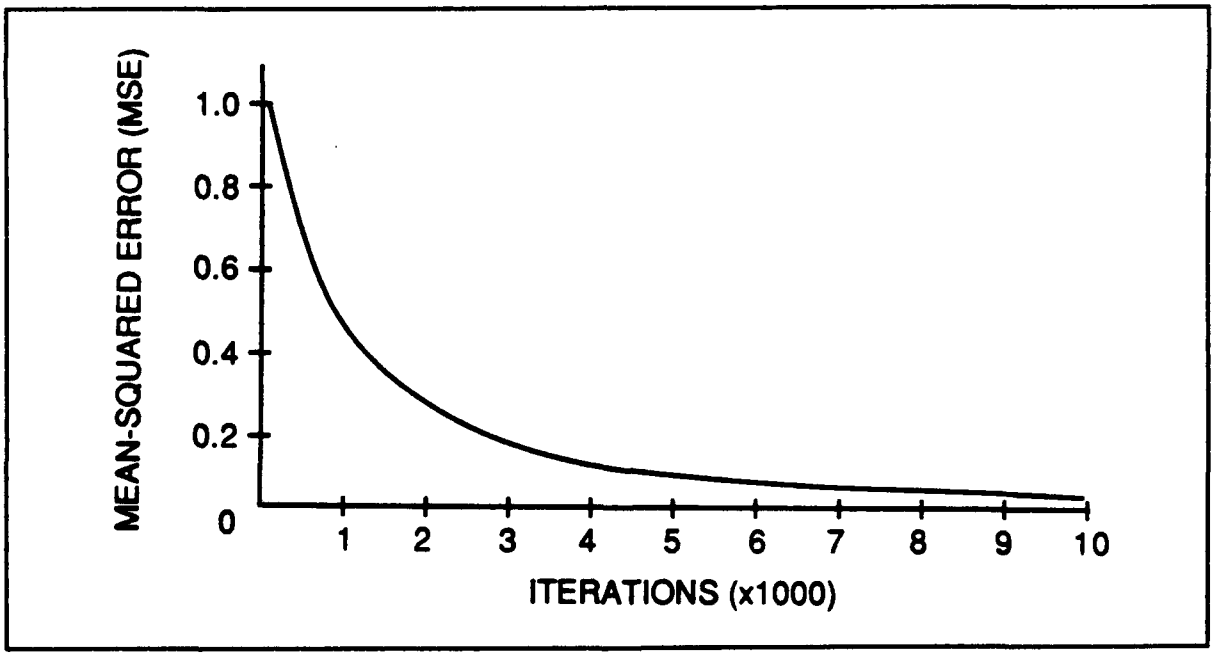


Figure 6. The Asymptotic Reduction of MSE as a Function of Network Training Iterations.

ANS Support Hardware and Software Description

In order to support the research and development efforts of this program, a specialized computation environment was developed which enables the rapid prototyping of neural network architectures and provides for effective visualization of the problem domain. This advanced computing environment is described in the following section.

ANS Processing Hardware

Neural network processing can place an extreme burden on conventional computer hardware, so specialized equipment in now being utilized to provide greater network size and processing speed capabilities. The development and implementation of the neural network code for the ACM Expert System has made use of a customized neural network processing board called the ANZA-Plus Neurocomputer from Hecht-Nielsen Neurocomputers (HNC). The ANZA-Plus coprocessor is part of an 80386-based computer system which is optimized for training and executing neural network software. A software development package for building and interfacing to various neural networks is included with the ANZA-Plus system. The ANZA-Plus coprocessor has a maximum combined capacity of 2.5 million processing elements and interconnections. Typical sizes of the combined number of PEs and interconnections for the ACM Expert System network are currently in the range of 500 to 1000, leaving plenty of room for future growth. The memory capacity on the board itself is 10 megabytes which allows for fairly large training sets. The processing speed for back-propagation training on the ANZA-Plus is 1.5 million interconnect updates per second. During run mode, the processing speed jumps to 6 million sustained interconnect updates per second. These speed and size limitations are easily outside any demands that a near-term realization of the ACM Expert System neural network might require, and it is anticipated that the ANZA-Plus will serve as an adequate development and delivery environment for future versions of this work.

The host computer for this program is a Zenith 386/16 system running under the DOS 3.31 operating system. The 80386 microprocessor in this machine operates at a 16 MHz clock rate and takes advantage of an 80387 coprocessor for math calculations. The Zenith contains a 360 kilobytes and 1.2 megabyte

flexible disk drives and an 80 megabyte internal hard disk. Internal memory consists of four megabytes of RAM. For display purposes, the Zenith is connected to a monochrome display and a Video Graphics Array (VGA) board and color monitor.

HNC Support Software

A wide array of development and interface software is provided by HNC for use with the ANZA Plus coprocessor. There are 17 predefined neural network paradigms in what is known as the Neurosoft library. Neurosoft is specialized ANS software which is designed specifically to run on the ANZA Plus board. All networks in the Neurosoft library have been tested and documented by HNC to decrease development time and risk. There are three levels of access available to the user during ANS development on the ANZA Plus system. At the highest level is a family of fully contained, rapid-prototyping tools called NetSet and ExploreNet which allow the user to quickly create and test a variety of networks using a "fill in the blanks" approach. ExploreNet is a windows-based, rapid development environment. Earlier versions of this product were utilized for some of BSED's initial ANS experiments. However, a high-level environment like ExploreNet necessarily entails a number of user restrictions. The ExploreNet system cannot be incorporated into user software nor can it be modified by the user, limiting its use for advanced neural network applications. The arrangement of the ANZA Plus hardware and software components is shown in Figure 7. The ExploreNet products interface to the rest of the system at the same level as the "user program" in the figure.

The next level of neural network interface software is known as the User Interface Subroutine Library (UISL), which provides the user with flexibility in accessing the Neurosoft paradigms. The UISL is a set of function calls which are embedded directly into the user's C language modules. As can be seen in Figure 7, the UISL is the central control routine for all network programming with the ANZA Plus. Though it requires more in the way of programming ability, the UISL provides the user with much more control over the design and development of the network and enables the network to be linked directly into the host software. Both the Lead Pursuit/Intercept system and the ACM Expert System are implemented through the use of the User Interface Subroutine Library. In these cases, the UISL has been used exclusively to

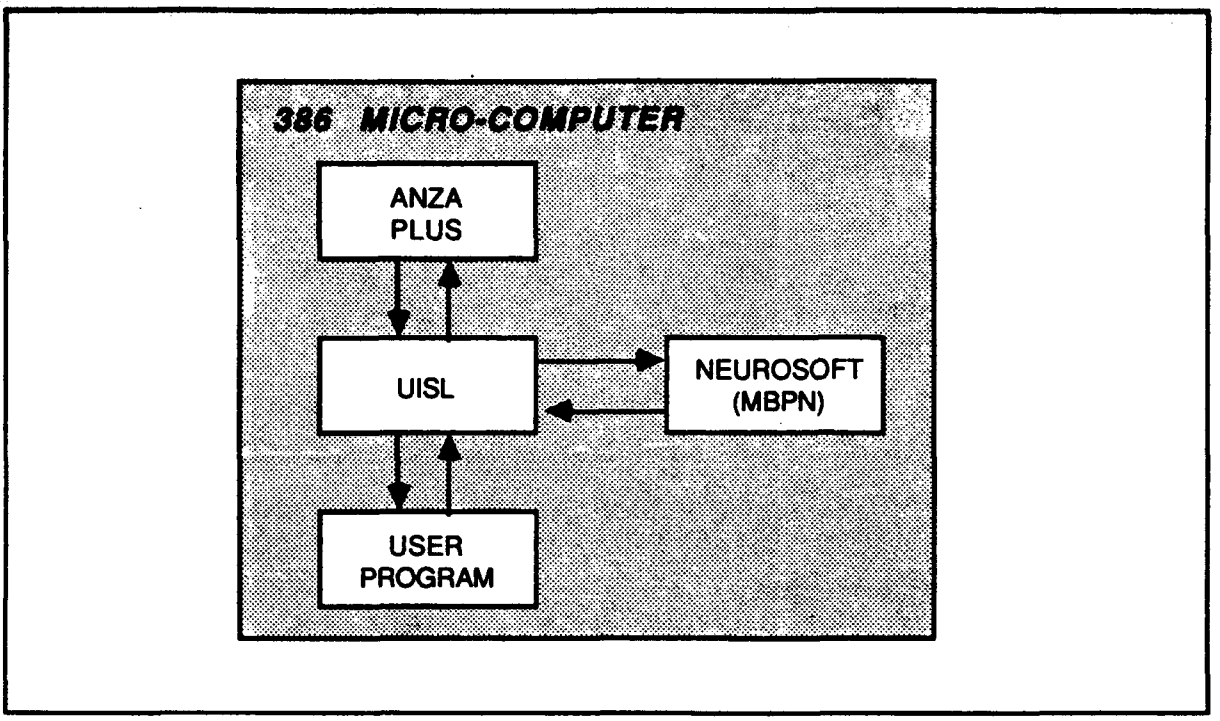


Figure 7. Arrangement of the ANZA Plus Hardware and Software components

access the Multilayer Back-Propagation Network package of the Neurosoft library.

The third level of interface capability allows the user to actually create his own neural network paradigms (or modify existing paradigms) by using a specialized network description language called AXON. With AXON, the user can exactly define the internal structure and update equations of a network which are then accessed from the host program via the UISL. Creating neural network code with AXON is analogous to adding a customized paradigm to the NeuroSoft library. BSED has used the AXON package to develop an enhanced MBPN with internal feedback from the hidden layers. This architecture is known as a **recurrent network** and is not available through the NeuroSoft package.

BSED ANS Software

In addition to primary ACM Expert System code and the various ANS experiments, various ANS software support tools for the design, development, and validation of neural networks were also developed. All software developed

under this program was written using the C programming language (Microsoft C Version 6.0). One of the fundamental tools created by BSED is a Neural Network Training System which allows an MBPN network designer to create a network for training or to continue the training of an existing network. The user is prompted for the various network definition parameters listed in Table 1, and the network is instantiated accordingly on the coprocessor. The training program then loads the specified training data into ANZA Plus memory and begins processing of the network. During training, the user is provided with a color display of the network structure, some of the training parameters, and a dynamic picture of how training is progressing. Figure 8 shows a typical display screen from the Neural Network Training System.

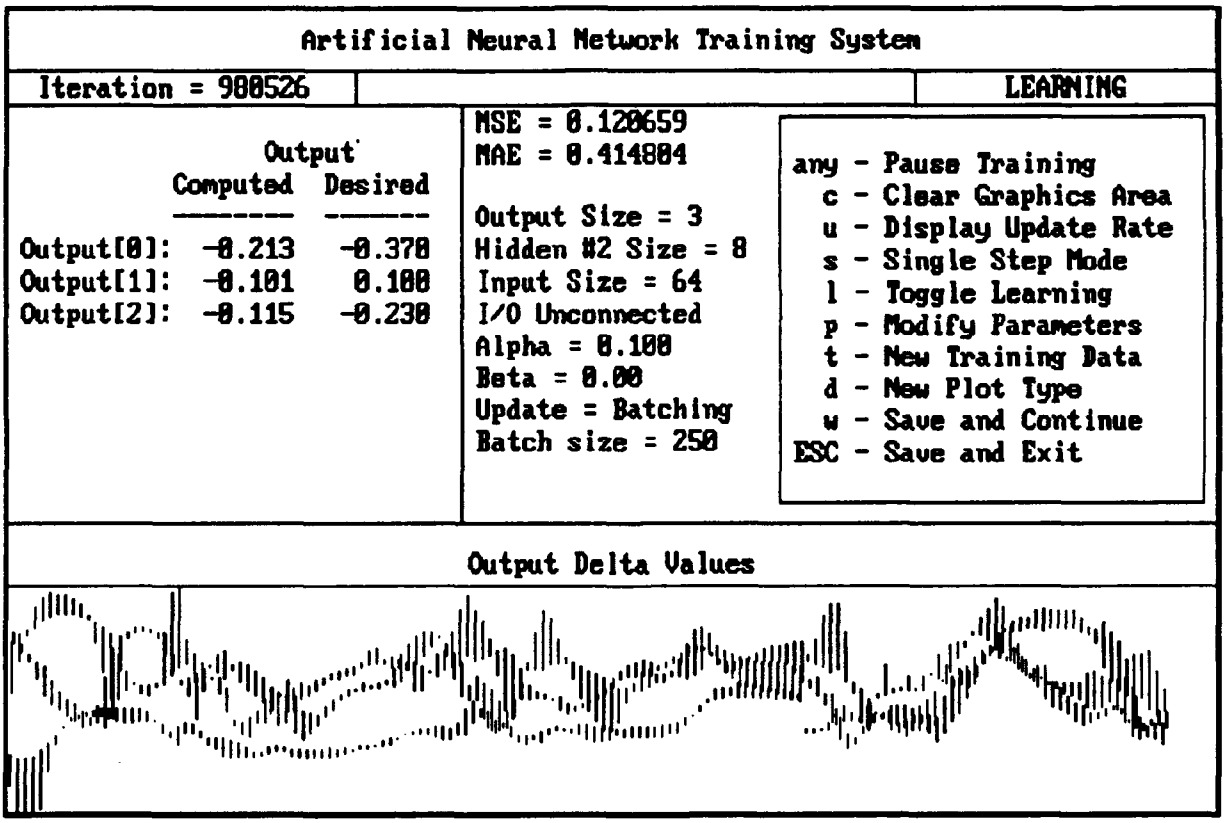


Figure 8. Neural Network Training System with Delta Output ValuePlot Mode Selected.

In Figure 8, the learning flag in the upper right corner indicates that the network is in the training mode. The iteration counter in the upper left corner indicates the number of training cycles that have been accumulated since this training session began. In the center left box, the numerical form

of the network-computed and target output values is shown. At the top of the center box are the Mean Squared and Absolute Error (MSE and MAE) values. In the center of the screen are the primary network structure and training parameter values. In this case, there are 64 input PEs, one layer of 8 hidden PEs, and an output layer of 3 PEs. Note that the HNC MBPN paradigm numbers the hidden layers in reverse order; therefore, the first hidden layer is designated layer #2, the next, layer #1, and the last, layer #0. In this example, since there is only one hidden layer, it is referred to as layer #2. In this case, input layer is not connected directly to the output layer. The learning rate, α, is set to 0.1, and the weights update mode is set to "Batching" with a batch size of 250.

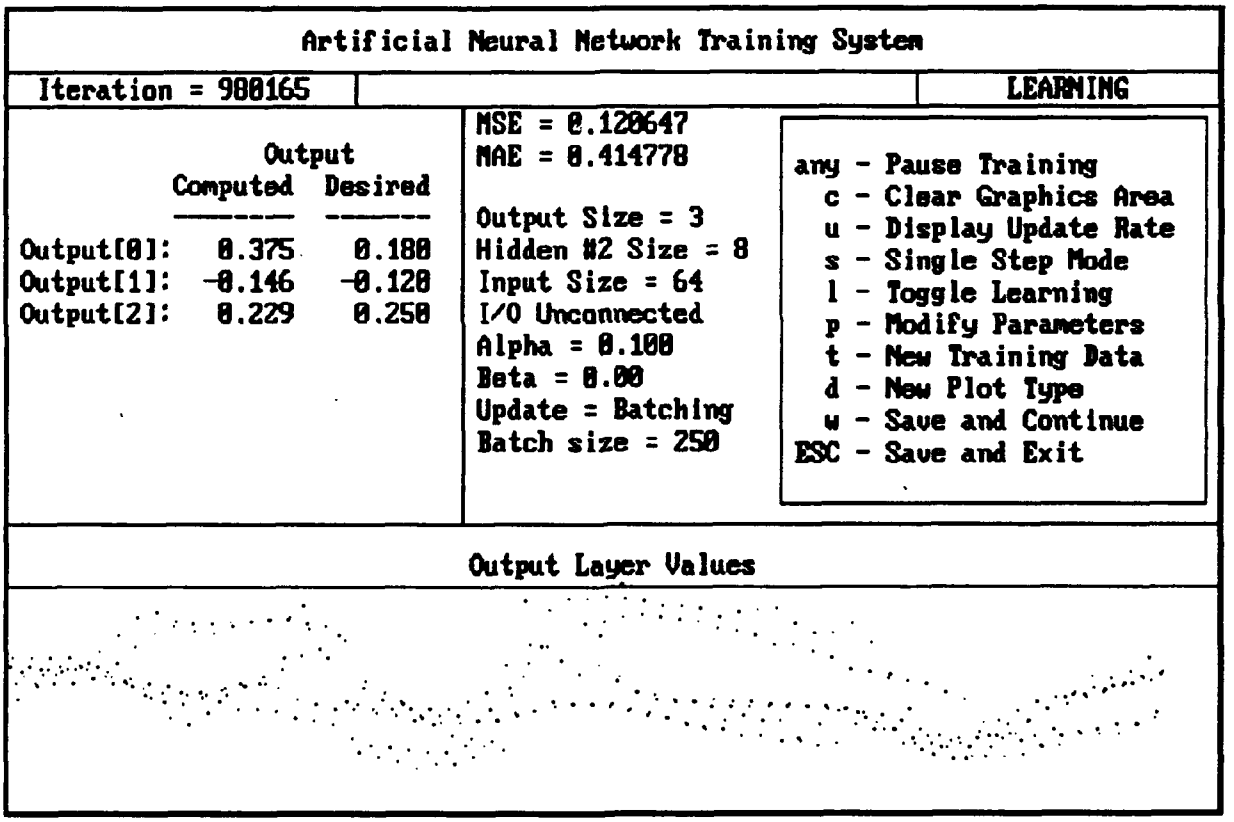


Figure 9. Neural Network Training System with Output Values Plot Selected.

At the bottom of the display screen is a window which provides a graphical representation of the output values over the duration of network training. The Neural Network Training System provides three plot modes for the graphics window. In Figure 8, the "Output Delta Values" mode has been selected. For each display update, a colored line is drawn between the

computed and target output value. The smaller the line, the closer the output value is to the desired target value. Figure 9 shows a similar screen, except that the "Output Values" plot mode is being used in the graphics area. This mode generates colored points for each of the computed and desired output values. In Figure 10, the "Error Values" plot mode has been used to display the values for MSE and MAE over the duration of network training. In all but the "Error Values" plot, the scale of the Y-axis in the graphics area corresponds to the range of the activation values of the output PEs. In the example shown, this is -1.0 at the bottom of the graphics area and +1.0 at the top. The "Error Values" plot is scaled each time it is selected so that the current maximum error value is the maximum Y value and the minimum Y value is zero. The X-axis of the graphics area represents the number of iterations. The display is updated every n iterations, where n is a user-selected value between one and ten thousand. Depending on the number of iterations between updates, the user can generate coarse or fine resolution displays in the graphics window. In Figures 8 and 9, the update rate has been set to update the display once every network iteration. In Figure 10, an update rate of one every hundred network iterations has been selected.

In each of the display figures, the box at center right is a run-time menu which provides the user with keyboard-selected options to clear the graphics area, change the display update rate, initiate a single step mode, turn learning on and off, modify the run-time training parameters, select a new data file for training, save the current weight structure, and select the desired output type for the graphics area. Typing any key other than those listed on the menu pauses the training process until another key is pressed. When the training process is complete (or at any time the user so desires) the training session may be terminated by hitting the escape key. At this point a prompt appears which asks for the name under which to save the current network weights and parameter definitions.

The network training system can be used in a "nontraining" mode to verify the performance of a given network by examining the output delta values and the network error displays. There are also optional file creation utilities which will build a history of MSE over time or record the activation levels of all PEs in the network for each display iteration. Network definition and training parameters are divided into two categories: **load-time** and **run-time**.

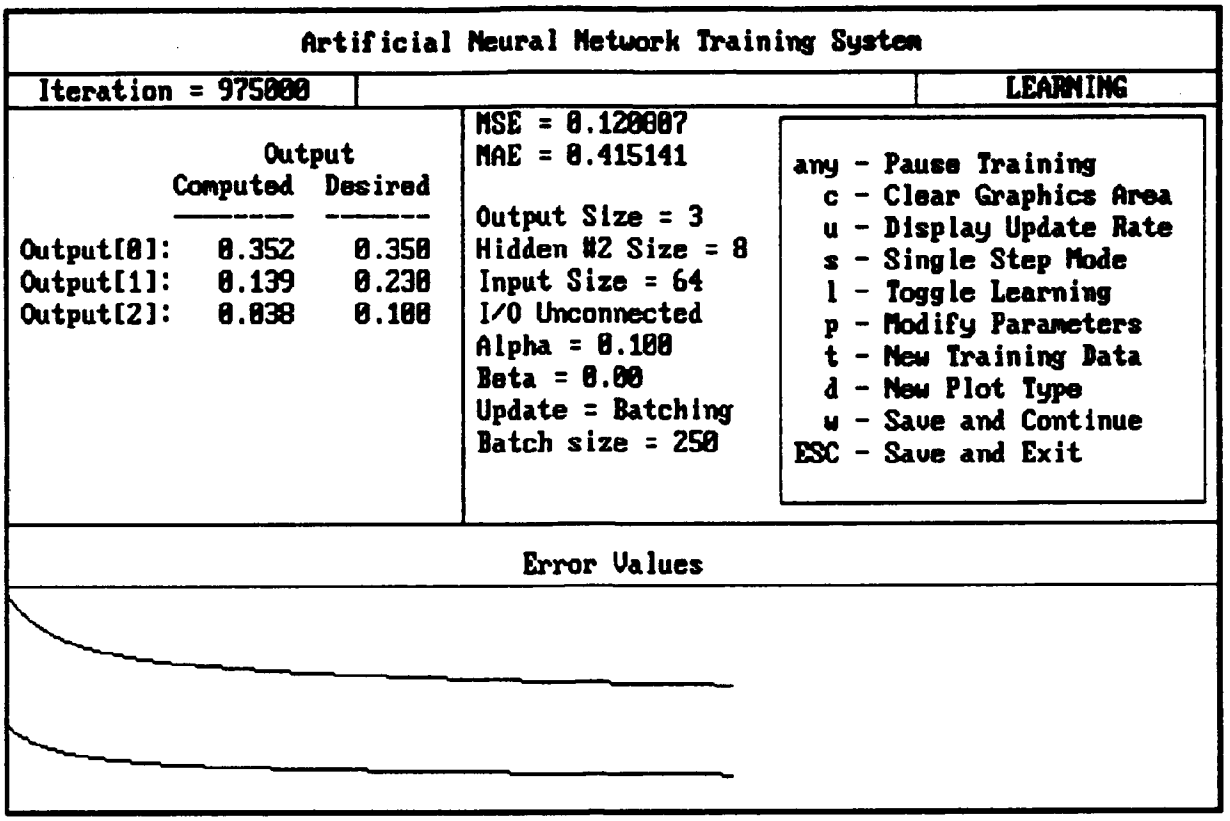


Figure 10. Neural Network Training System with Statistics Mode Selected.

Load-time parameters are defined before the network is instantiated and may not be modified thereafter. A sample screen from the MBPN Training System for entering load-time parameters is shown in Figure 11. The run-time parameters, shown in Figure 12, may be modified by the user during the training process.

```
                Modify Load-Time Constants

    Output Layer Size [3] ?
    Number of Hidden Layers [2] ? 1
    Size of Hidden Layer #2 [10] ? 8
    Input Layer Size [64] ?
    Random Seed [17264] ? 12345
    Initial Weight Max [1.00] ? 0.2
    Wts Update Allowed: 0=N, 1=B, 2=S, 3=S/B [3] ?
    Connect Inputs: 1=TRUE, 0=FALSE [0] ?
    Function Class: 0=LGS, 1=TLIN, 2=GAUSS, 3=ATAN, 4=ATANH [0] ?
    Steepness of Slope [1.00] ?
    Upper Limit [1.00] ?
    Lower Limit [-1.00] ?
```

Figure 11. Load-Time Constants Modification

```
                Modify Run-Time Constants

Output Alpha [.1000] ?
Hidden Alpha #2 [.6000] ?
Wts Update Allowed = 3
Wts Update Actually Used: 0=N, 1=B, 2=S, 3=S/B [0] ? 1
Batch Size [0] ? 250
```

Figure 12. Run-Time Constants Modification

Another tool used during the development of neural networks is the Weights Analysis Utility. When a network has been trained and saved using the Neural Network Training System, the training parameters and the current weight values are stored in two disk files. These files can then be used to produce a readable profile of the network's structure, training conditions, and weights. The Weights Analysis Utility allows the user to quickly examine or compare the details of any saved network. It provides a complete breakdown of each weighted connection in each layer of the network.

PRELIMINARY NEURAL NETWORK EXPERIMENTS AND RESULTS

Introduction

To develop an in-depth understanding of neural network behavior and build expertise in applying Artificial Neural Systems techniques to experts systems issues, BSED adopted an evolutionary development approach. Such an approach starts with individual elements of the full problem, scales them down to their simplest form, and explores and builds upon these relatively simple components. When the individual elements are understood in isolation, they are recombined to create successively more complex systems which lead to the working model. The fundamental elements of the ACM Expert System involve the representation of changing angles, ranges, velocities, and the transformation of these data into reasonable flight path commands. The following experiments demonstrate how these elements might be handled in a neural network representation and how the various individual elements may be combined. They also point out many interesting details of how neural networks behave when

attempting to learn the kinds of mappings involved in the simulation of ACM decision making.

The first set of experiments was the creation of a simple three-layer network to determine the proximity an input angle to a fixed angle. This series of experiments established the capability of neural networks to successfully learn the mapping of non-monotonic functions as found in aircraft angle geometry. Having examined the domain of angles, the next set of experiments involved a network which provides slant range as output based upon (X, Y) coordinate input data. The results of these two experiments were then combined to create a simple Tactical Situation Classifier network. This network takes antenna train angle (ATA), target aspect angle (TAA), and range as input and produces two output values. One output indicates the level of threat and, the other, the potential for kill under the current relative geometry.

The next step in the evolution of the ACM Expert System was the development of a network which simulates the behavior of a relatively simple lead pursuit and target intercept maneuver algorithm. Building directly on the experience gained from the earlier network experiments, the Lead Pursuit/Intercept (LPI) demonstration provided a direct point of departure for development of the ACM Expert System's neural network architecture. Additionally, by allowing a direct, objective comparison of a neural network and an algorithmic solution, much was learned about the generalization power of neural network implementations. A full description and the results of the Lead Pursuit/Intercept Simulation are presented later in this report. By upgrading specific modules of the LPI system, the prototype ACM Expert System was created. A diagram of how the ACM Neural Network has evolved is shown in Figure 13.

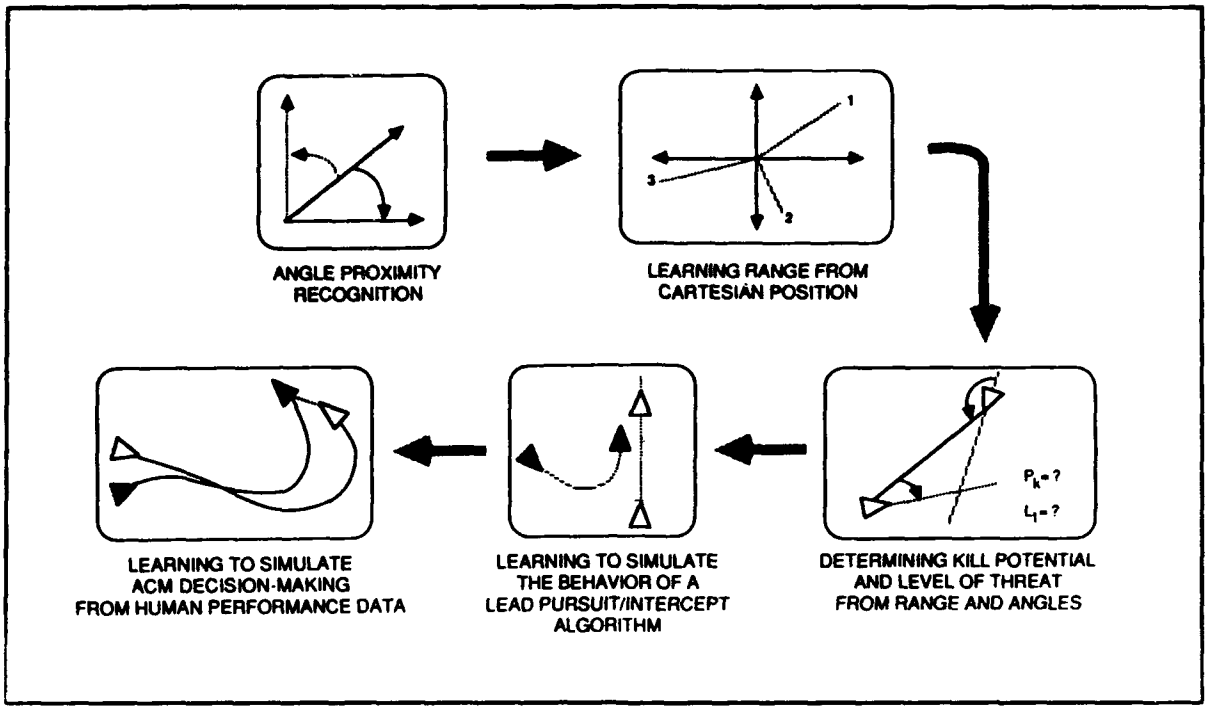


Figure 13. Evolution of the ACM Neural Network System.

Angle Proximity Calculator

The first series of experiments to be carried out on the ANZA Plus coprocessor addressed the ability of simple back-propagation networks to represent angle proximity relationships. Specifically, researchers were interested to see how well an ANS system could represent the unnatural mapping between angle values and processing element (PE) activation values. For instance, if angles from 0 to 360 degrees are represented by PE activations from 0.0 to 1.0, the network must learn that the angles of 1 and 359 degrees are relatively proximate even though their respective PE activations are maximally distant. This anomaly is diagrammed in Figure 14, where it can be seen that pairs of angles with similar proximity values are not necessarily represented by similar activation differences. The small angle between 1 and 359 degrees and between 1 and 3 degrees has a size of 2 degrees in both cases. However, the difference of the activation levels for each pair of angles is not the same. So how can activation levels be used to code for proximity? This is an example of a problem where the association between input and output

data is not inherent in the data itself and must be represented by abstractions within the neural network's hidden layer.

A theoretical neural network to solve this angle/proximity relationship might consist of a single input PE which represents angles from 0 to 360 degrees, a single output unit which produces a proximity value between 0 and 1, and a hidden layer of optimal size (see Figure 15). In this case, the network would be trained to associate input angles with their appropriate proximity to 0 degrees, where an output of 0 would indicate the angle was maximally distant from 0 degrees (180 degrees) and an output value of 1 would signify maximal proximity to 0 degrees. Associated proximity values for each angle in this configuration are given by the equation:

$$\text{proximity} = \text{ABS}(1 - \text{angle}/180). \qquad (11)$$

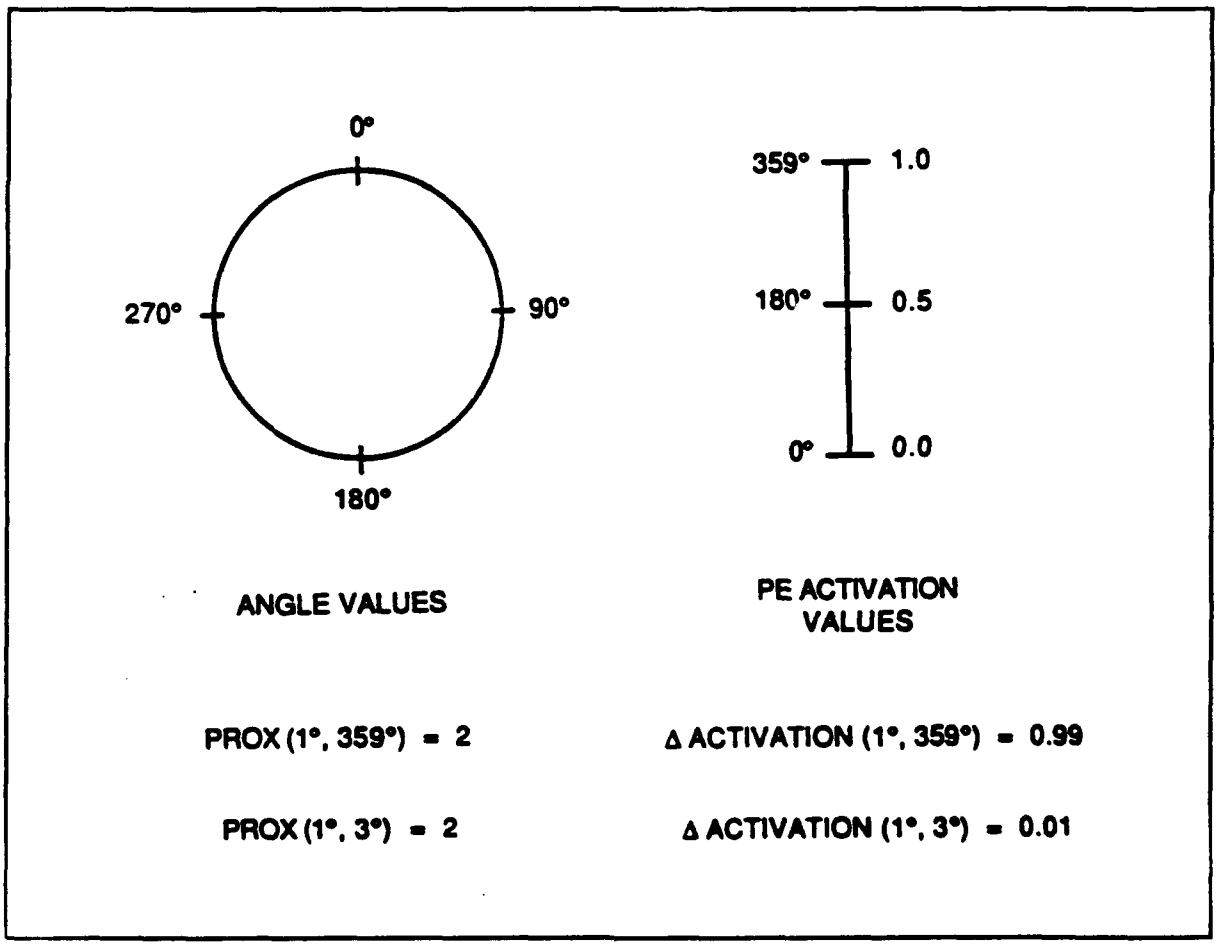


Figure 14. Mapping Angle Proximity Values to PE Activation Levels Presents a Challenge in a Neural Network Representation

This mapping is potentially difficult for a neural network because the monotonically increasing angle values are being mapped to nonmonotonic proximity values.

Networks with the above structure were tested on the ANZA Plus system using the NetSet (an early version of ExploreNet) development package. It was determined that the presence of a single PE in the input and output layers does not provide enough interconnections to create a stable network, and therefore, the fidelity of the signals generated by each PE is limited. For these reasons, an additional PE was added to the input and output layers, and

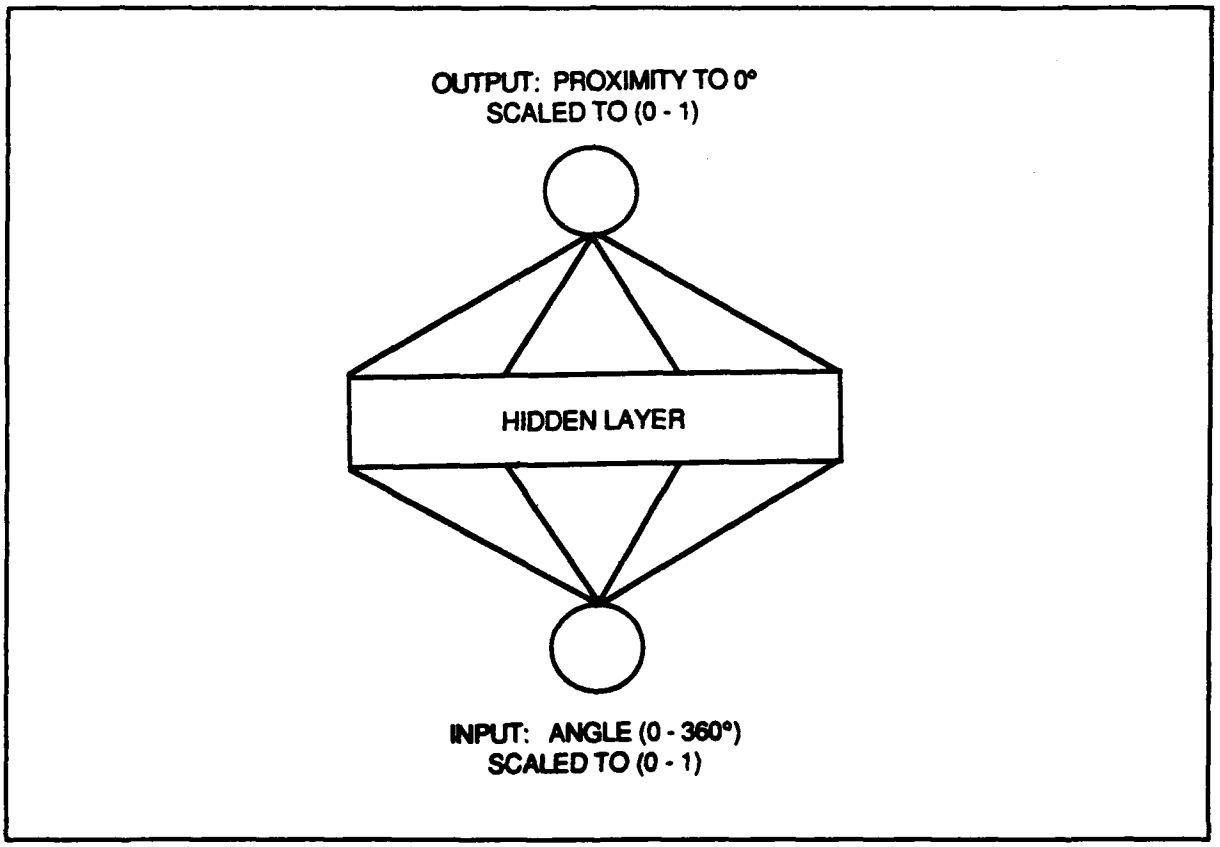


Figure 15. Theoretical Network Structure to Represent the Angle Proximity Problem

the input and output values were encoded to be divided evenly between the two units of each layer. For example, an experiment was carried out using the network structure shown in Figure 16. In this experiment, called EXPDA, the input angle is divided so that values less than 180 degrees are scaled between 0.0 and 1.0 and represented by input PE #1 while input PE #2 remains at zero.

Angles between 180 and 359 degrees drive input PE #1 to its maximum activation level while the angle value is scaled between 0 and 1 and applied to input unit #2. Similarly, proximity of the input angle to 0 degrees is represented on the output layer by assigning values between 0 and 0.5 to output unit #1 and values between 0.5 and 1.0 to output unit #2. The EXPDA network contains five units in the hidden layer.

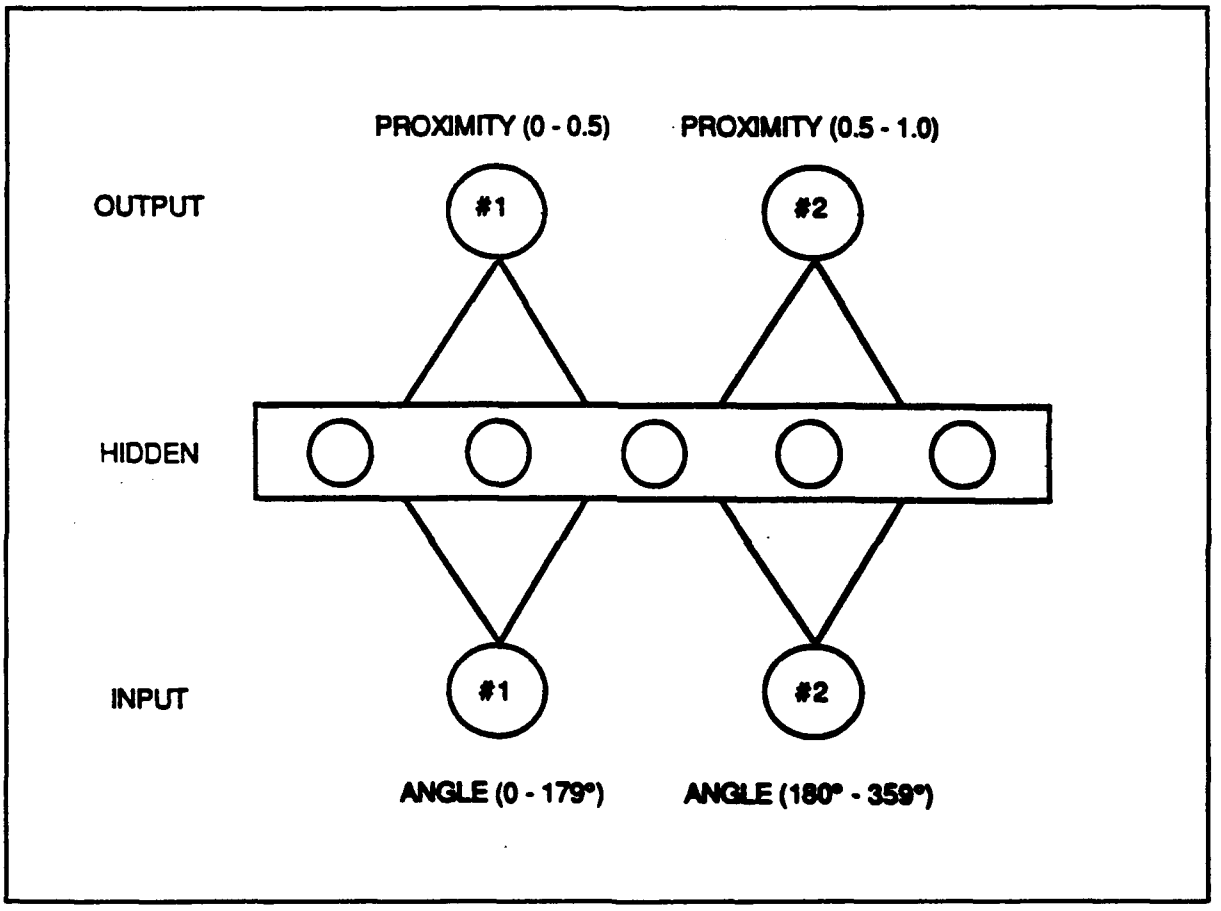


Figure 16. Neural Network Structure for the EXPDA Angle Proximity Experiment.

Training of the EXPDA network was carried out using four sets of data. The first set contains input angles and their corresponding output proximities for angles at 45 degree intervals. Subsequent training data sets contain finer angle intervals at 30, 15, and finally 1 degree spacings. Training begins with the coarse data and proceeds to the finer data as the network begins to learn the relationship between angle and proximity to 0 degrees. A sigmoidal thresholding law was used, and the batching and momentum terms of the back-propagation learning algorithms were disabled. Before training, the

interconnection weights were randomized using a weight initialization seed of 123 and a weight initialization range of ±0.5. The learning rate constant, α, was set initially to 2.0 and decreased as training progressed to a final fine tuning value of 0.1. At each stage of training, the weight adjustment process was carried out for approximately 20,000 iterations. When the training process was complete, the network's capability was tested using the 1 degree data in a single production run. The results of the EXPDA experiment are shown in Figures 17 and 18. Figure 18 shows the normalized output values (scaled between 0 and 1) from the neural network for each output element. These are the encoded values of output as they are represented on the network's output layer. The outputs are plotted as a function input angle from 0 to 360 degrees. Both the network-computed and the actual target data for each output PE are displayed, resulting in four separate curves. Notice that the network's output departs from the target data at very specific points on the curves. In Figure 18, the unbroken curve is generated by decoding and recombining the activity levels of the output layer PEs. This curve represents the network-computed proximity values as a function of input angle. The second curve, depicted by the broken line, represents the actual or target proximity values as a function of angle.

As can be seen in Figures 17 and 18, the network was quite successful at learning the angle proximity representation, though there is still a certain amount of error in performance. Though further training may shave a bit more error from the network's performance, much of the exhibited error is due to an apparent inability of the learning process to faithfully track sudden changes in the direction or magnitude of the training data. In Figure 17, the directional changes can be seen as sharp points of inflection in the target data. Output PE #1 has inflection points at 90, 180, and 270 degrees, while PE #2 has inflection points at 0, 90, and 270 degrees. These inflection points are due partly to the inherent structure of the problem, as shown in Figure 19, and partly due to the way in which proximity values are divided among the two output units. Experiment EXPDA also contains a discontinuity in the data representation where the input values jump from 359 degrees back to 0 degrees.

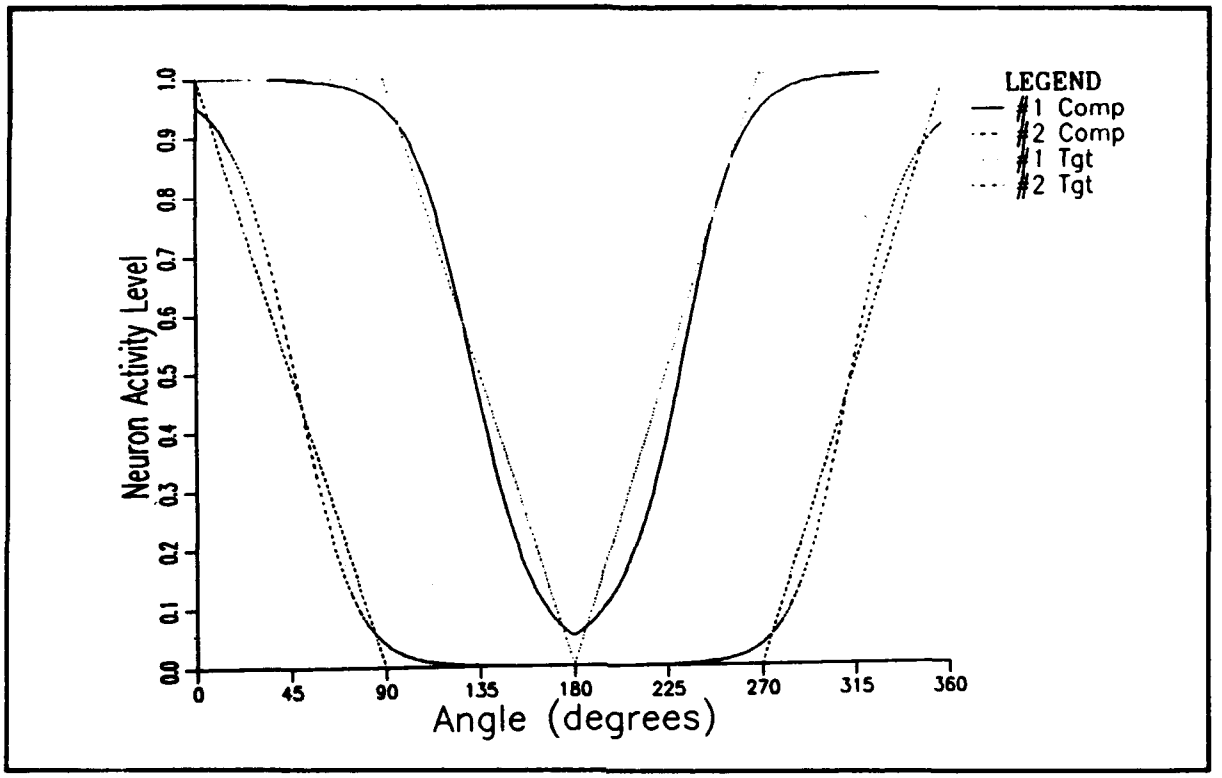


Figure 17. Output PE Activity Levels as a Function of Input Angle in the EXPDA Network

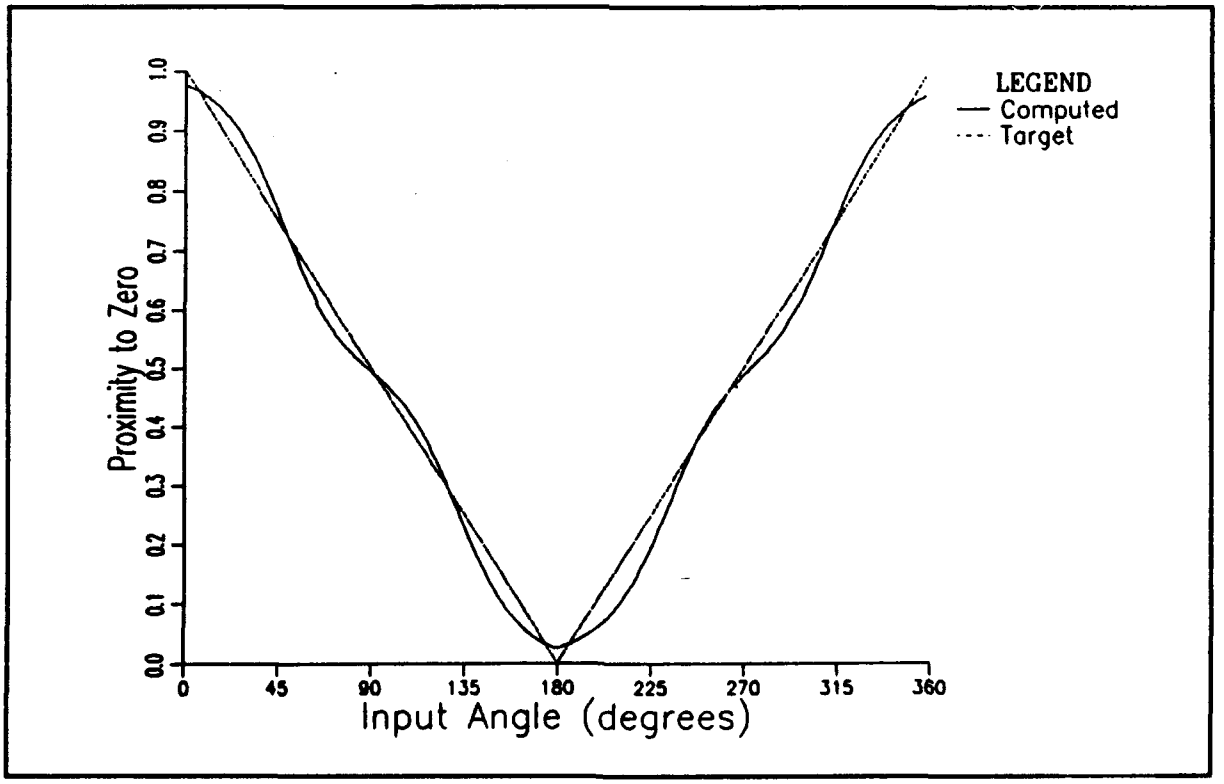


Figure 18. Proximity Value Output as a Function of Input Angle in the EXPDA Network

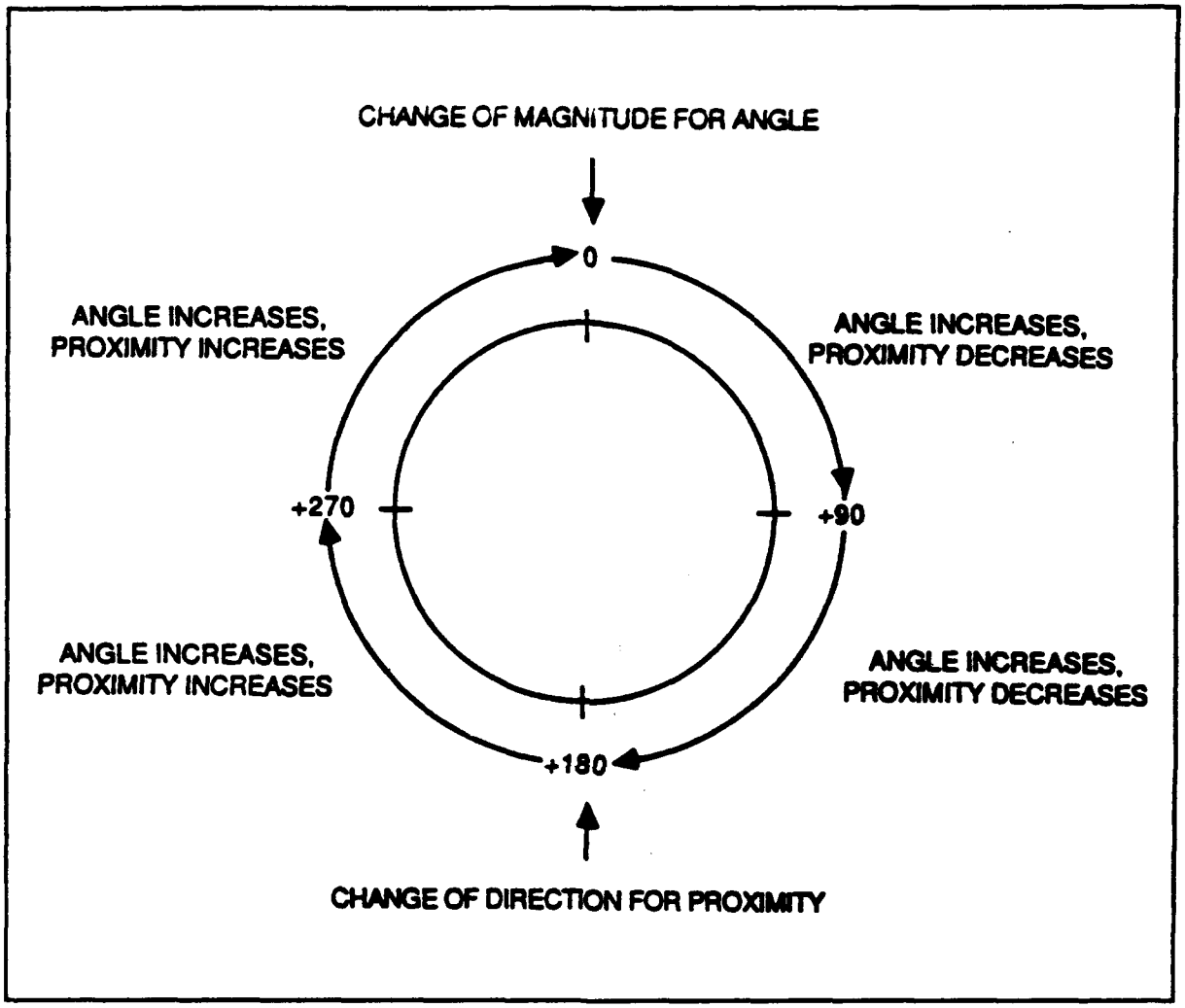


Figure 19. Points of Inflection in the Representation of Angles and Proximities to Zero Degrees in the EXPDA Experiment.

A similar experiment, called EXP2, was conducted with a different representation of angle at the input layer and a new target proximity of +90 degrees, rather than the zero degree target of EXPDA. For the input, instead of dividing a 0 to 360 degree angle among the two units, PE #2 coded for angles between 0 and 180 degrees while PE #1 indicated positive (1.0 activation level) or negative (0.0 activation level) angle values depending on whether the angle was on the right or left side of the circle. This arrangement circumvents the change in magnitude problem of the 0 to 360 case, but increases the number of inflection points (one at each quadrant) as can be seen in Figure 20. The network connection structure of the EXP2 experiment is identical to that of the EXPDA structure.

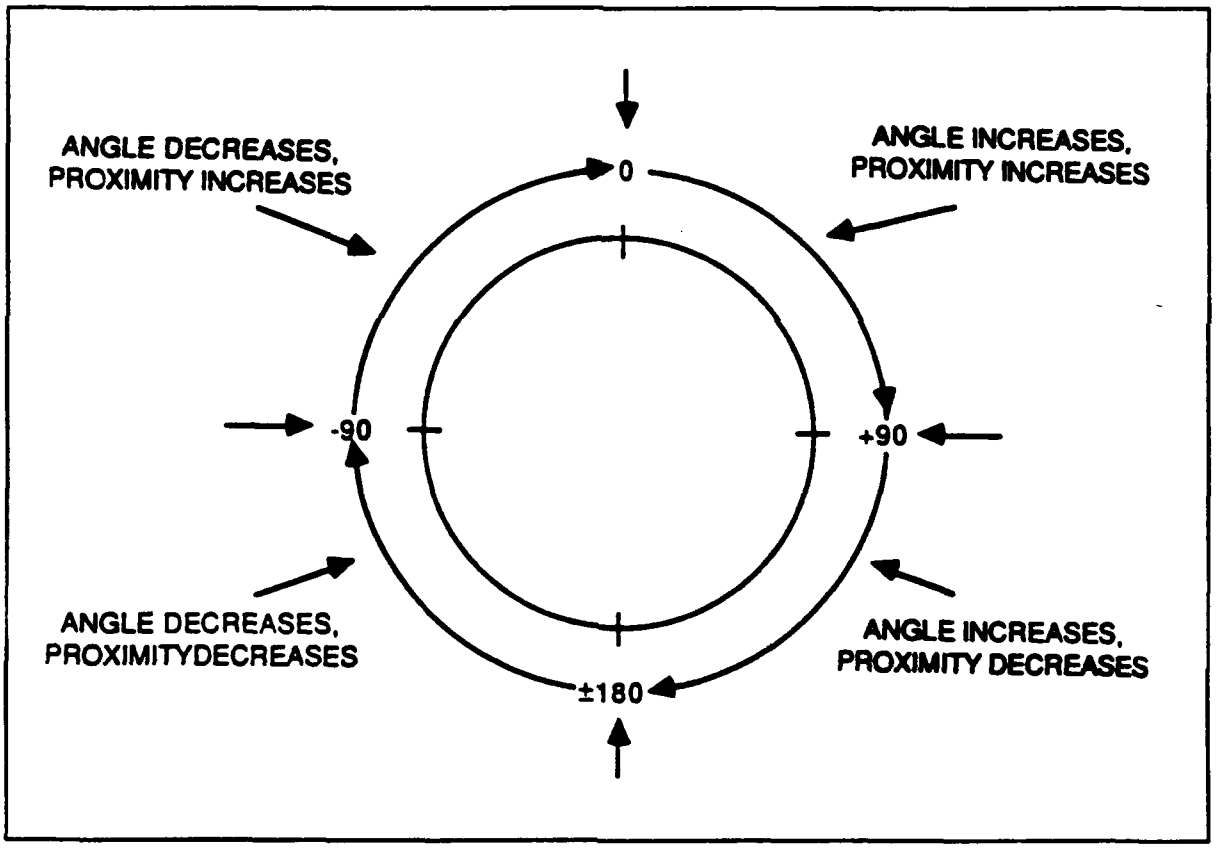


Figure 20. Points of Inflection in the Representation of Angles and Proximities to +90 Degrees in the EXP2 Experiment.

Using the same training procedure and initial conditions as were used in the EXPDA experiment, the EXP2 network produced very similar performance results. The computed and target values of the network's output units are graphed together in Figure 21. The actual proximity versus angle data for the

same set of one-degree sample data is shown in Figure 22. To get a numerical comparison of mapping accuracy, the target and computed output values for the EXP2 and EXPDA experiments were subjected to an absolute error test using the following equation:

$$\text{absolute error} = \sum_{i}^{360} \text{abs}(tgt_i - comp_i) \qquad (12)$$

where tgt_i is the target proximity output desired for a given input angle, i, and $comp_i$ is the network-computed proximity value for the same input angle. When the average absolute errors in proximity output for EXP2 and EXPDA are compared, it is evident that both networks were capable of achieving close to the same results through quite different representations of input angle. The EXPDA network has an average absolute error of 0.026 over 360 one-degree samples, while the EXP2 network produced an average absolute error of 0.017 over the same range. This seems to indicate that the back-propagation learning process is capable of overcoming various kinds and numbers of discontinuities or inflection points in mapping from input to output data.

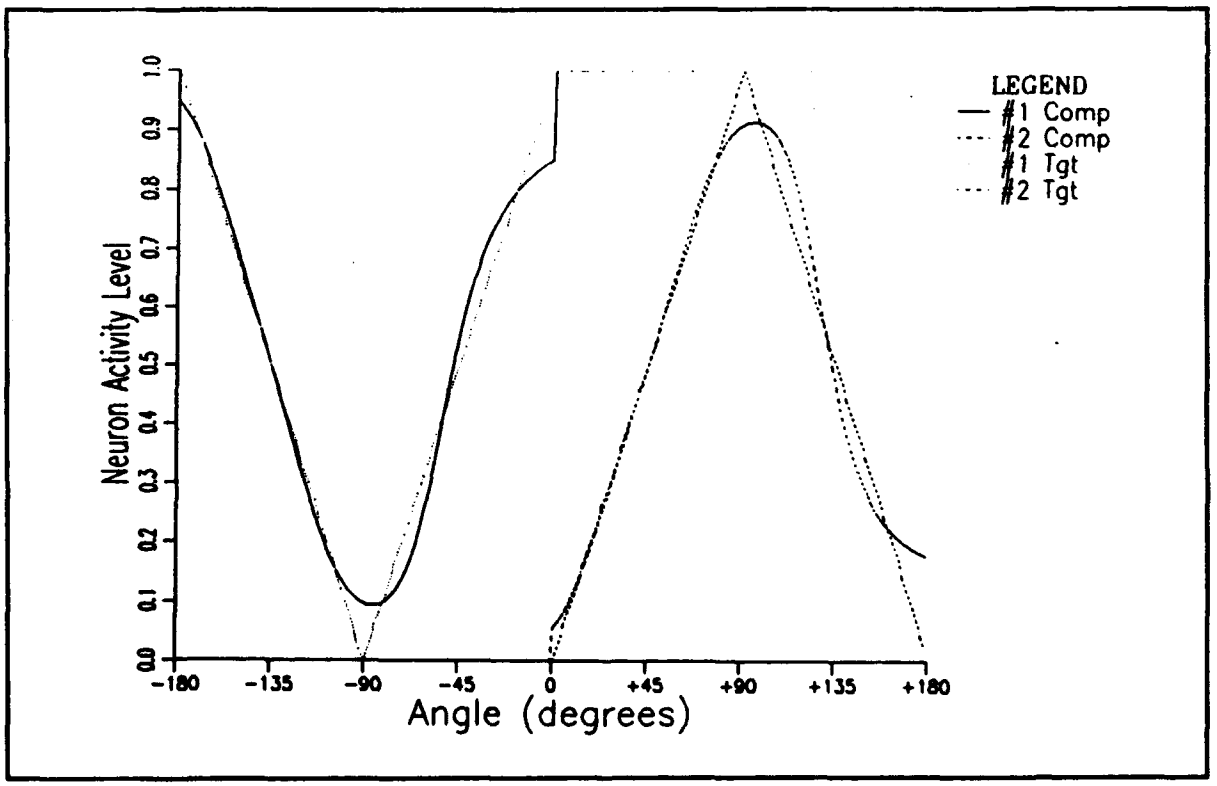


Figure 21. Output PE Activity Levels as a Function of Input Angle in the EXP2 Experiment.

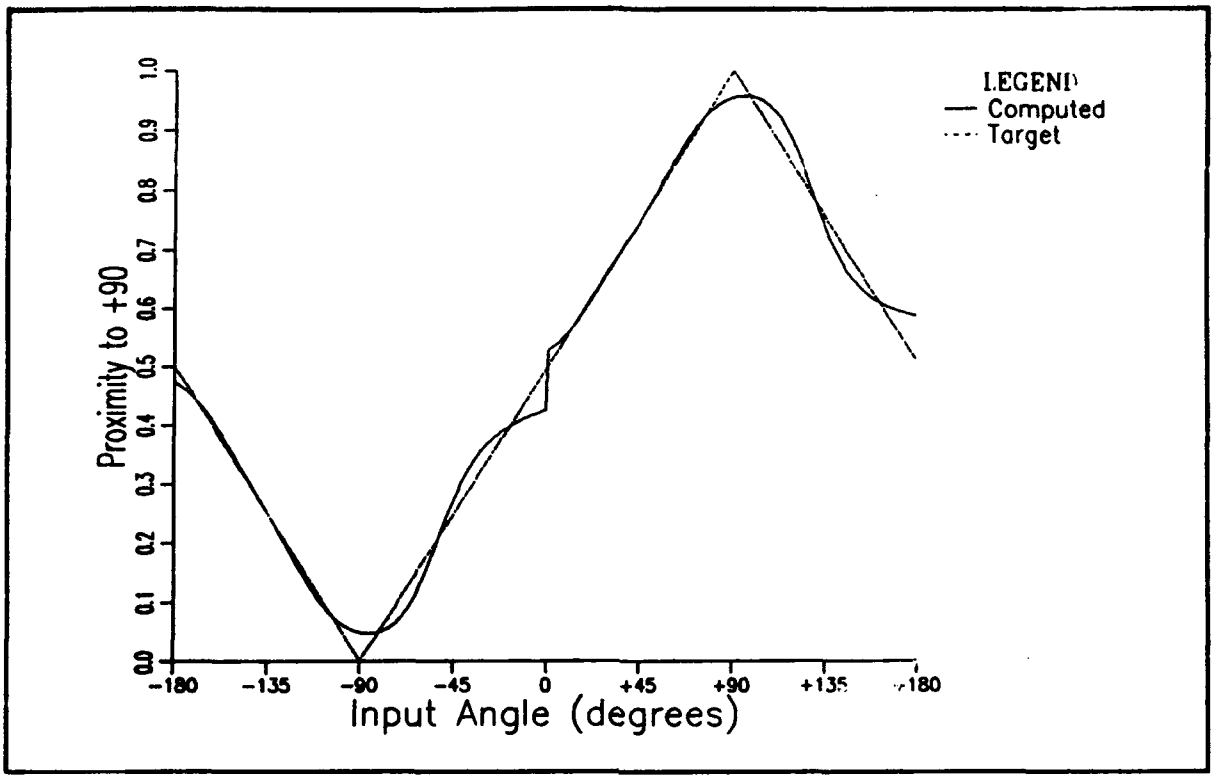


Figure 22. *Proximity Value Output as a Function of Input Angle in the EXP2 Experiment.*

Though the average error rate for the networks is relatively low, the error is not evenly distributed along the entire range of data. The computed data diverges at certain points while matching perfectly at others. As mentioned before, the inflection points in the target data seem to be influencing the network's inability to perfectly track the target data. In an effort to determine whether the sequential nature of the training data presentation has an influence on the final network performance, a third experiment, called EXPR, was devised to recreate EXP2 with nonsequential training data. It was assumed that training took on a certain character due to the fact that, in certain sequences of sequential data, the weight changes for $angle_{n+1}$ are not much different from the changes for $angle_n$. However, when the data reaches a point of inflection, like the edge of a quadrant in EXP2, the weight changes must alter their direction and thus, the computed data temporarily depart from the target data. To test the influence of these monotonic training data sequences, the 45, 30, 15, and 1 degree training data, which associate angles to target proximities, were randomized in terms of

their order of presentation to the developing network. The results of the EXPR test are shown in Figures 23 and 24.

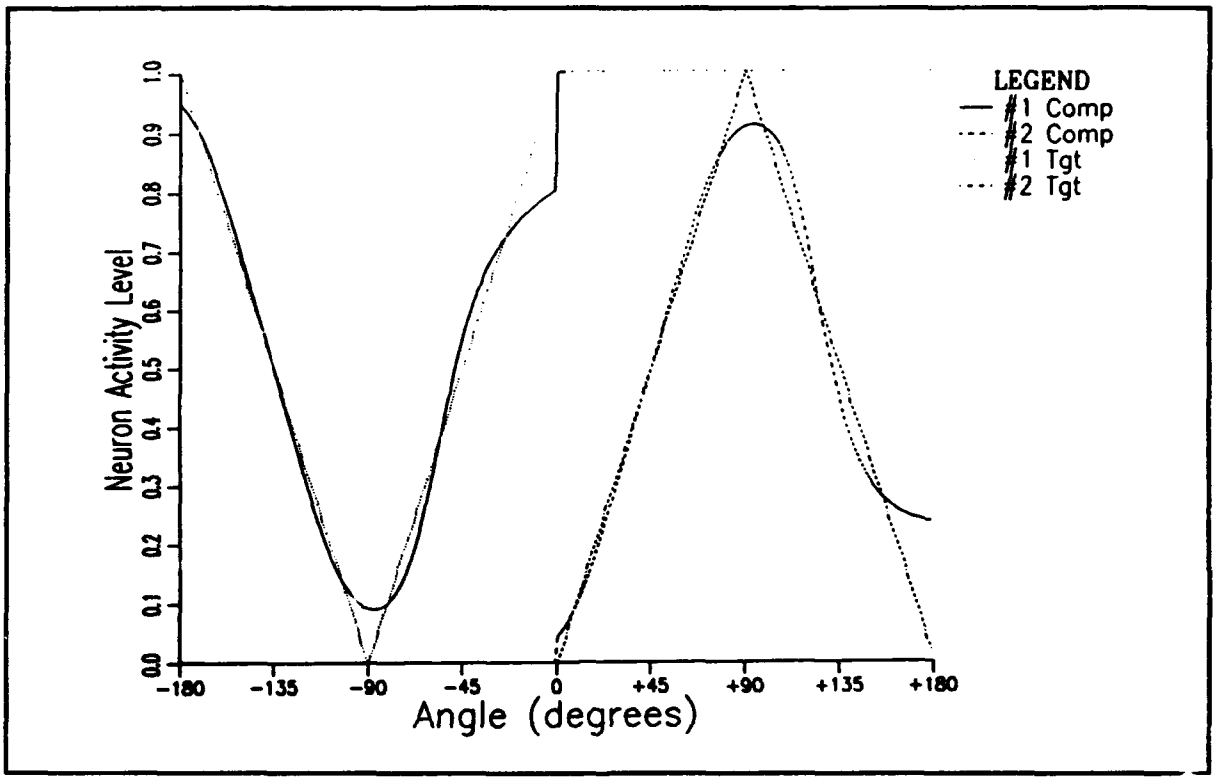


Figure 23. Output PE Activity Levels as a Function of Input Angle in the EXPR Experiment.

Graphically, there seems to be little difference between the performance of the sequentially trained EXP2 network and the randomly trained EXPR network over the 360 degrees of test data, and indeed, the average absolute error between target and computed proximity values for EXPR is 0.019, essentially the same as the sequential case. This indicates that, at least for this problem, the role of data presentation sequence in the training of basic back-propagation networks is not significant to the final weight structure of the network.

The primary conclusion to be drawn from these simple excursions is that the representation of a single data value with more than a single processing element should be avoided since the resultant introduction of inflection points caused by this representation produces difficulties in the neural network learning process. Certain data may require some division of their

external representation among more than one PE, but as long as the data is continuous and monotonic, it should be represented by a single PE to maintain its continuous and monotonic nature. If a given neural network requires that the accuracy of its input and output elements be increased, the number of hidden units must be increased, not the number of input or output units. In the previous experiments, the number of input and output elements was increased to overcome the inherent instability of very small networks, but this is not an issue when the representations are scaled up to model more complex problems requiring a greater number of input and output PEs.

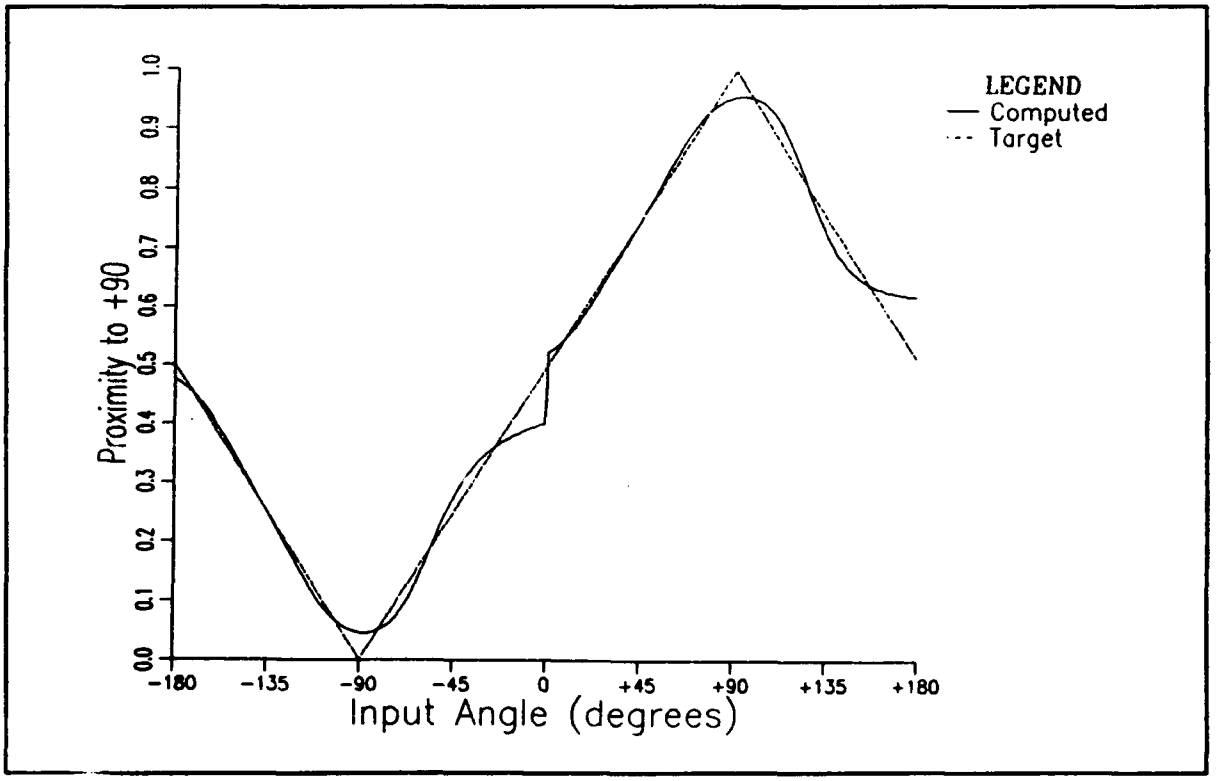


Figure 24. Proximity Value Output as a Function of Input Angle in the EXPR Experiment.

A general result of the Angle Proximity experiments is that neural networks of this type are able to generalize quite easily. For example, if the network in the EXP2 case is trained exclusively on the 30-degree training data, it will eventually learn to provide accurate proximity results with an average absolute error of around 0.007 when tested with the same 30-degree data. When this network is then tested with the 1-degree data set, it also provides reasonable proximity output for those angle values upon which it was not trained. The average absolute error for a network so trained is 0.030 for

360 one-degree data samples. This compares favorably with the error rate of 0.017 produced by the EXP2 network which had the benefit of being exposed to the 1-degree data during training. The network which has seen only 30-degree data during training learns to generalize its responses to provide approximate output values for input angles which fall between the angles it has been trained on.

The concept of a solution space is quite easy to visualize in the Angle Proximity networks. It is clearly represented by the broken lines in Figures 18, 22, and 24. Since the solution space is simply a two-dimensional map that relates a single angle value with a single proximity value, it becomes a trivial procedure to select a uniform distribution of training samples which span the entire space. This is precisely what was done in generating the training files for the EXP2 and similar experiments. The only question for the network designer is what resolution to use; in this case the finest samples were made up of one-degree increments, but even 30-degree samples provided a good mapping for network training because the data was uniformly distributed across the entire space. In later experiments, including the ACM Expert System, where the input and output representations are more complex, the exact nature of the solution space is much more difficult to define.

The mechanics of "training out" the error in a given network are dependent, in part, on the learning rate, the length of training, and the type of learning equations used during the back-propagation process. In general, the network will learn a rough approximation of the desired mapping rather quickly, but the reduction of mean squared error (MSE) is an asymptotic procedure which may require many iterations to reap a relatively small increase in network performance, see Figure 6.

Range From Cartesian Coordinates

The second set of neural network experiments is similar to the Angle Proximity tests, but in this case, a neural network was created to map cartesian points to a slant range value from a fixed point. Given a 100x100 mile grid with a fixed point at position (50,50), the network is to determine slant ranges from the fixed point to various points in the grid. The network structure used to accomplish this task is shown in Figure 25. The input units

represent the x and y coordinates between 0 and 100 miles, and the output units represent the range to (x,y) from (50,50). All values are normalized to a 0 to 1 scale, and the range value is divided between the two output PEs, as was done with proximity values in the previous experiments. The training data were generated by randomly selecting x_2 and y_2 values and calculating range with the equation:

$$\text{range} = \text{sqrt}\ (x_2 - x_1)^2 + (y_2 - y_1)^2 \qquad (13)$$

where (x_1,y_1) is the point (50,50). The range network was trained using the same parameters as were used in the Angle Proximity networks.

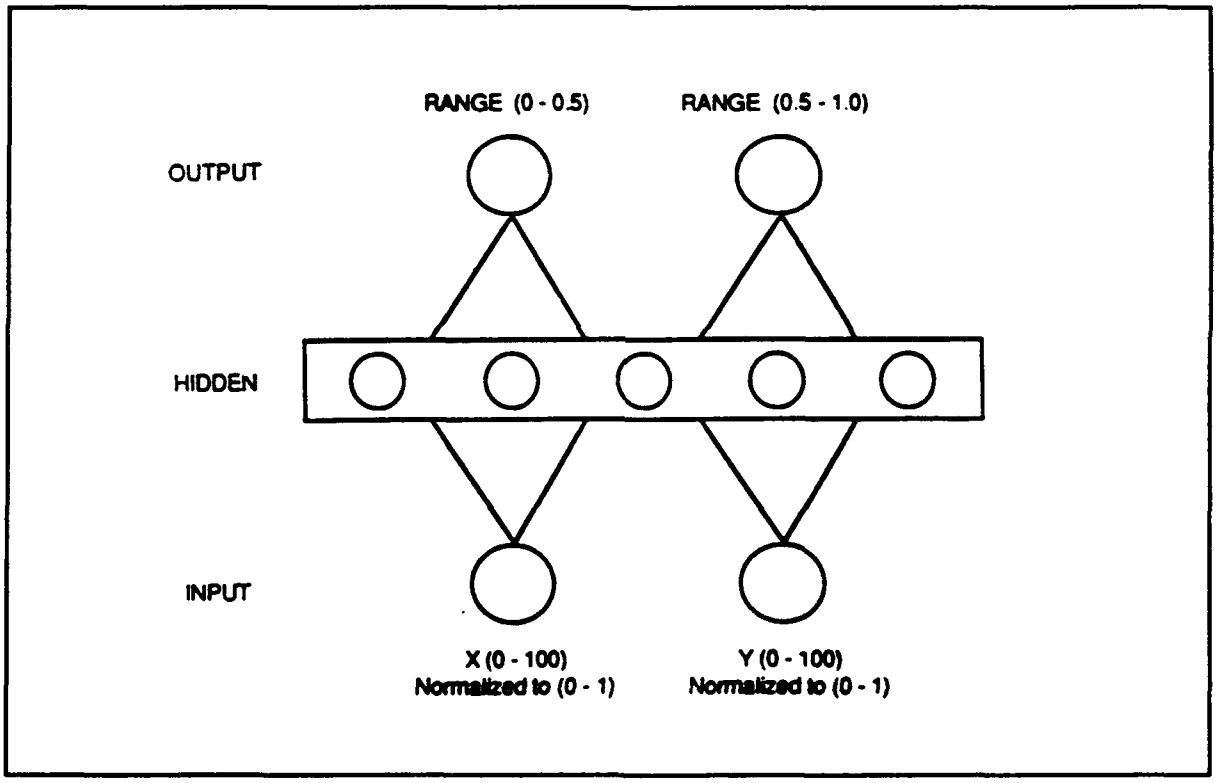


Figure 25. Neural Network Structure to Map Cartesian Coordinates to Range Values.

After training, the level of network performance was found to be similar to that obtained in the previous experiments. Performance was tested by creating a data set based on a series of points which lie on the diagonal from (0,0) to (100,100). The network-computed and target activation levels of the output units for the test data are shown in Figure 26. The actual range

versus position data (network-computed and target) are plotted in Figure 27. Points of inflection in the input and output data for the range experiment are evident in the network's uneven mapping between computed and target data. As can be seen in Figure 26, the point of greatest deviation from the target data occurs closest to the fixed point, (50,50). This points out a deficiency in the representation of range. To generate a high-fidelity representation of range from the minimum to maximum range values, a neural network may require two range input PEs, one for coarse representation and one that is sensitive

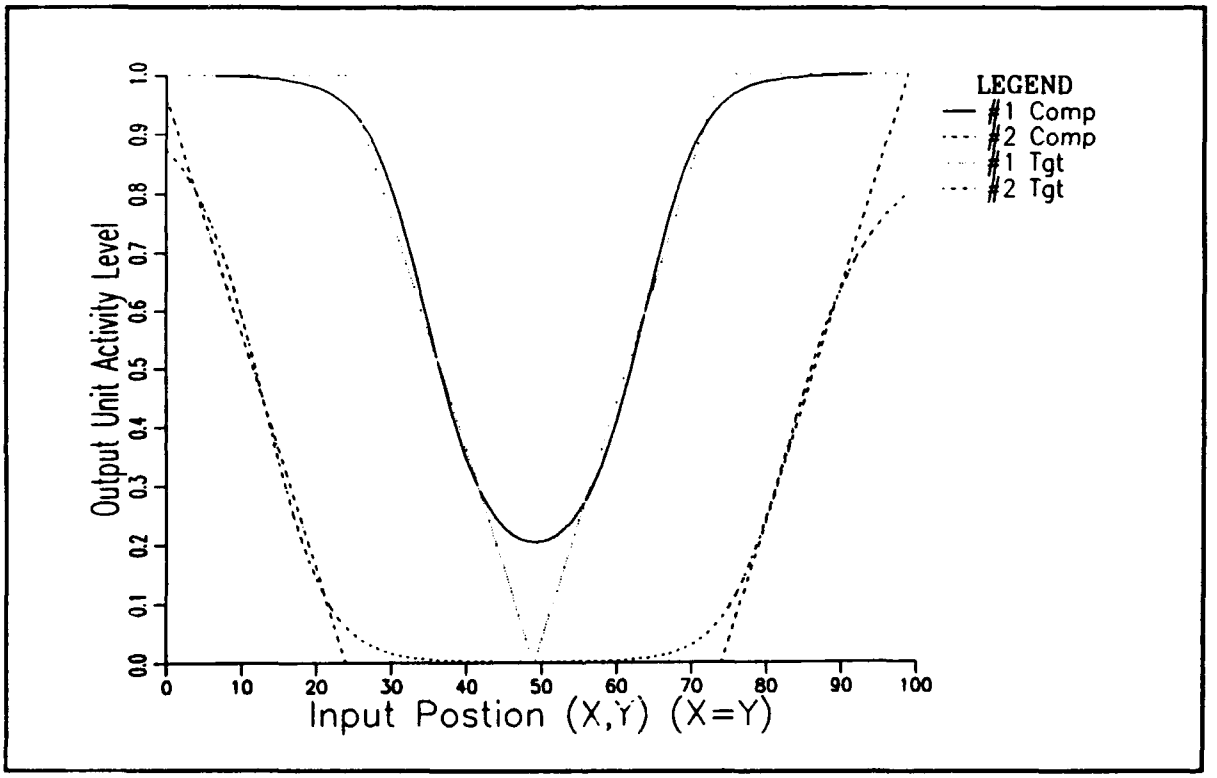


Figure 26. Output PE Activity Levels as a Function of Input Coordinates along the Diagonal from (0,0) in the Range from Coordinates Experiment.

only to close ranges. The average error over the 100 random test data points, and for the diagonal test data, is 1.47 miles for the trained range network.

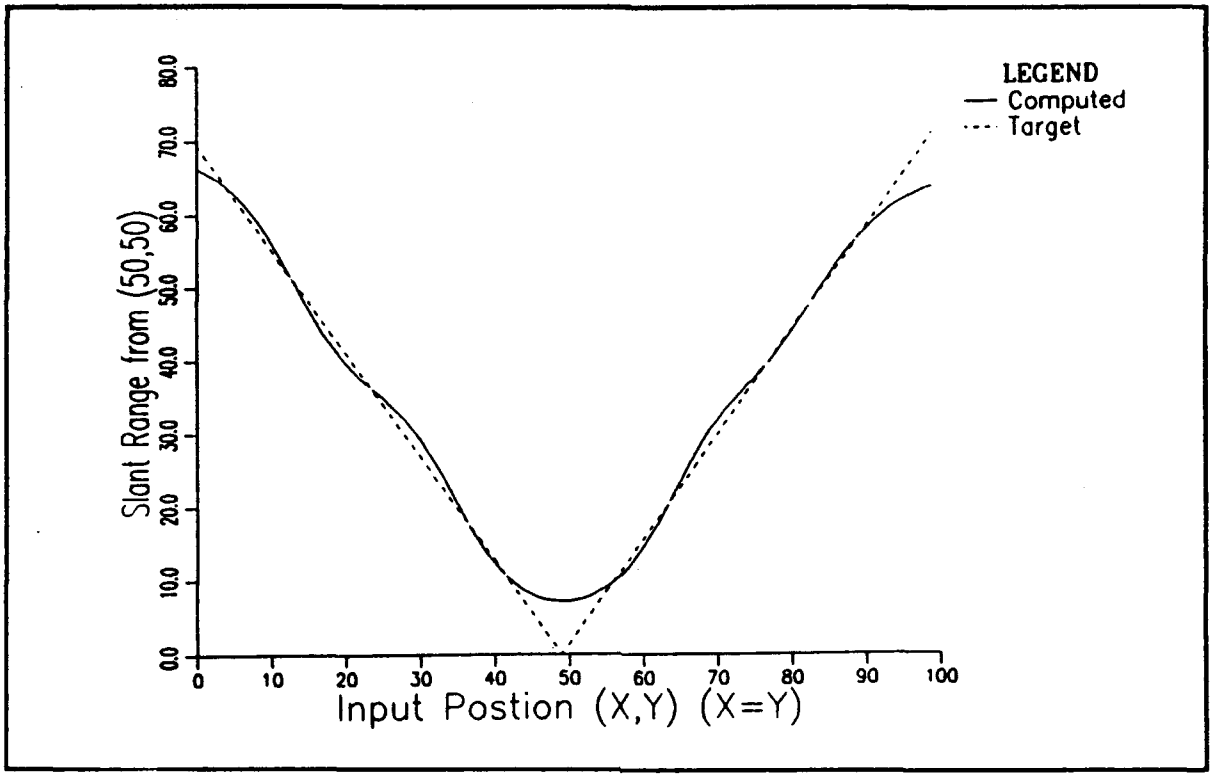


Figure 27. Range Output Values as a Function of Input Coordinates along the Diagonal from (0,0) in the Range from Coordinates Experiment.

Tactical Situation Classifier

An obvious next step in the evolution of the ACM Expert System would be to combine the network performance for representing angle and range relationships into a single network. This is precisely what the Tactical Situation Classifier (TSC) Network was designed to do. The TSC neural network takes antenna train angle (ATA), target aspect angle (TAA), and range to target as inputs and produces a value between 0 and 100 (100 being maximum) for both level of threat, (L_t), and potential for kill, (P_k), under the current "tactical situation". The training of this network was based on two simple functions of angles and distance which produce the level of threat and potential for kill values. The form of the functions used to generate the training file are shown below:

$$L_t = f(\text{ Range, ATA, TAA }) \qquad (14)$$

where L_t is inversely proportional to Range and TAA, and proportional to ATA.

$$P_k = g(\text{ Range, ATA, TAA })$$ (15)

where P_k is inversely proportional to Range and ATA, and proportional to TAA.

Whether the equations are entirely valid is not as significant as that they are consistent in the output values they generate. A random set of 500 input conditions were used to create the network training file. Random sampling was used in an effort to get a uniform distribution of training samples across the comparatively larger-solution space. The input and output values were scaled between 0 and 1 before presentation to the network. Figure 28 shows the TSC network structure. After about 20,000 iterations, requiring less than 30 seconds of training time, the network exhibited excellent performance on a random set of test data, producing an MSE of 0.0003 over the entire test set. The TSC experiment clearly shows that a neural network is capable of learning a mapping for the combined elements of the ACM domain and is able to generalize its solution from a limited set of examples. It also demonstrates that such a mapping can be accomplished very quickly for relatively simple networks.

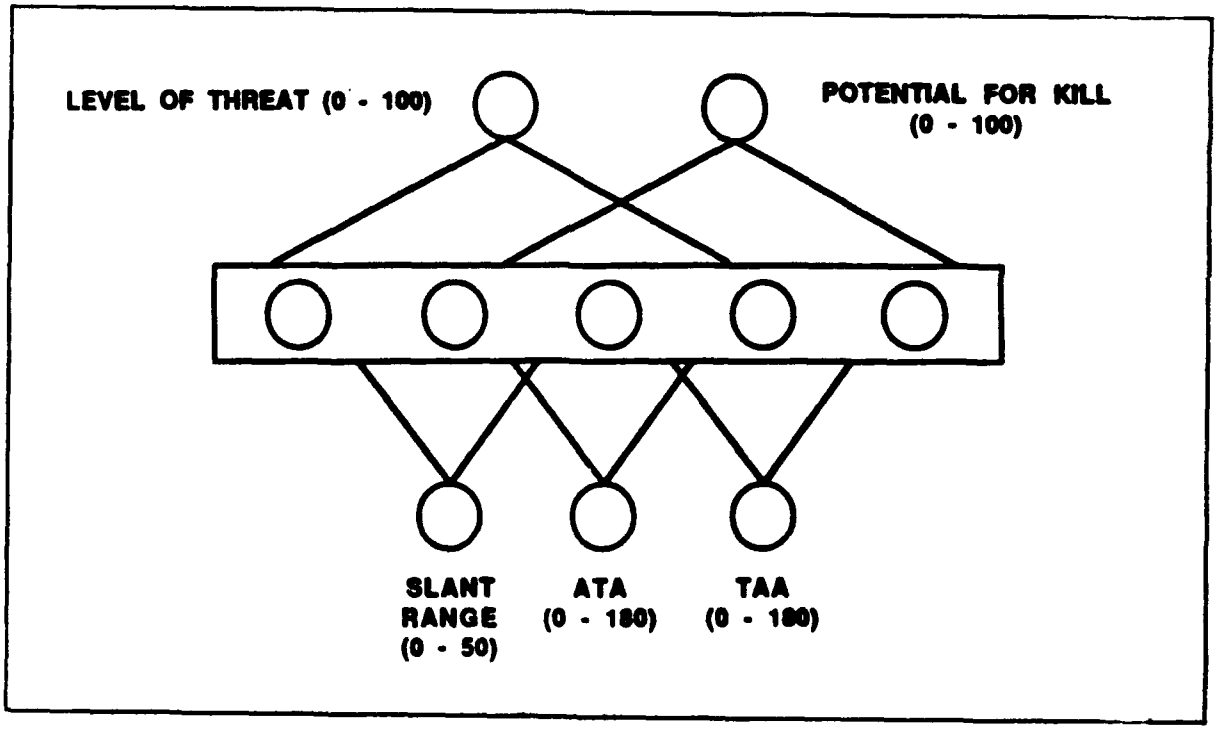


Figure 28. Neural Network Structure used to Represent the Tactical Situation Classifier.

Lead Pursuit/Intercept Simulation

After establishing that neural network representations were capable of handling the simple angle and range relationships of the previous experiments, the next step was to combine all the underlying elements of the problem into a single neural network which could operate on flight profile and maneuver data. This was intended to serve as a scaled-down prototype of the ACM Expert System. The goal was to design and implement a neural network which could simulate the behavior of a simple flight path generator. Though this network system would not be based on human performance data, and would involve only a simple, restricted set of maneuvers, it would provide a relevant domain within which to create a working model of an ACM neural network development environment. The major difference between the Lead Pursuit/Intercept (LPI) system and the ACM Expert System is that the LPI demonstration is based on flight data generated by an algorithm while the ACM neural network will be trained on actual pilot performance data. Despite this difference, the LPI system serves as a logical point of departure toward the target system for the following reasons. First, it provides an operational testbed for the development and validation of the various software components that are required for the ACM Expert System. By using a relatively simple maneuver profile for the problem domain, the individual software modules could be validated and integrated in a scaled-down and workable environment, but still be exercised using data that is similar in form to the expected ACM data. The use of maneuver data generated by an algorithm allowed neural networks to be trained with data that is consistent, easily generated, and quickly validated. Furthermore, the LPI software modules were designed specifically to be upgraded into the actual components of the ACM Expert System software. Finally, it was desirable to create an interim system which would provide a visual presentation of the ANS solution process for demonstration purposes. The LPI demonstration met all of these requirements and served as a major evolution toward the development of the ACM Expert System.

The first step was to put together an algorithm which would successfully fly a lead pursuit profile followed by an intercept maneuver for any initial relative geometry of two aircraft. The basic operating parameters of this algorithm are outlined in Figure 29. The algorithm was then used to generate time history data of various flight profiles which could serve as training

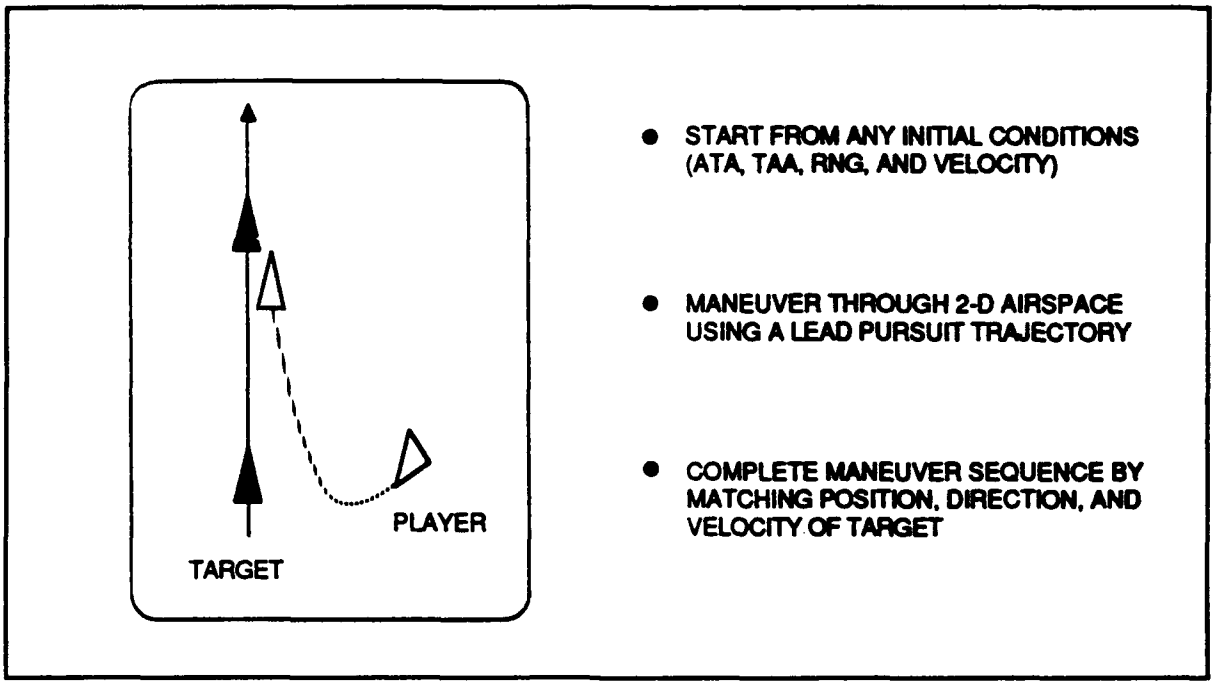


Figure 29. The Basic Operating Parameters of the Lead Pursuit/Intercept Algorithm.

data for a neural network. If successful, the neural network would learn by example to accomplish the lead pursuit and intercept procedure without knowing anything about how the algorithm worked. The network was expected to duplicate the performance of the algorithm simply by being exposed to examples of algorithm's behavior. The framework for this experiment was to be a demonstration system which plotted the algorithm-generated and network-generated flight profiles simultaneously so that their relative performance could be easily compared. The key element of this test would be the neural network's ability to create a generalized mapping of input to output since only a very small subset of all possible conditions would be used for training. An overview of the LPI neural network simulation process is shown in Figure 30.

Description of the Algorithm

The Lead Pursuit/Intercept Algorithm is actually comprised of two separate algorithms. The Lead Pursuit algorithm controls the flight path until the pursuer is close and behind the target. At that point, the

Intercept algorithm matches the target's location and airspeed to within a specified tolerance. For the LPI ANS demonstration, the tolerances require that the pursuer be within 30 meters of the target with less than 10 meters per second (m/s) of velocity difference for a successful intercept and termination of the profile. Acceleration commands to maneuver the pursuit aircraft are generated based on the current relative geometry of the two aircraft at an update rate of one per second.

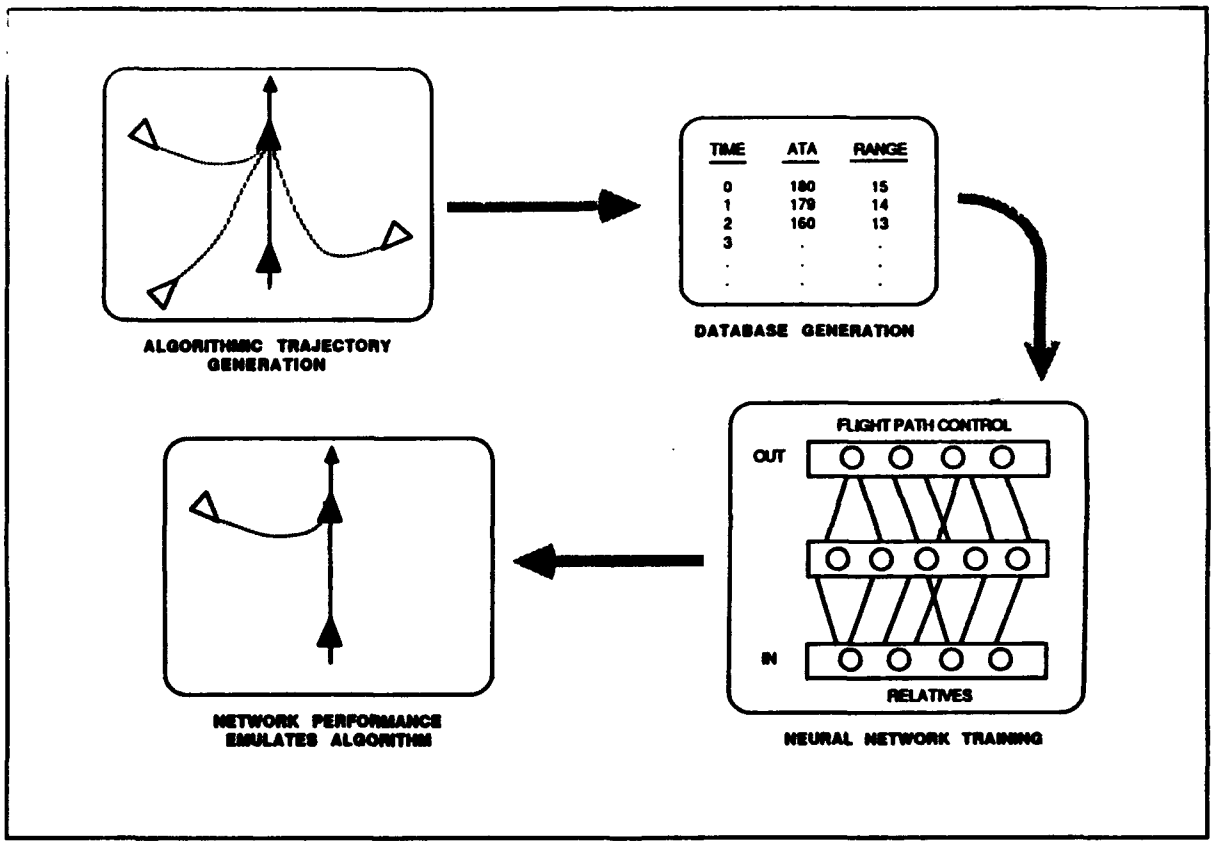


Figure 30. The Lead Pursuit/Intercept Neural Network Development Process.

The general flow of control in the LPI algorithm can be seen in Figure 31. The target and pursuer aircraft are assigned initial values for position, orientation, and airspeed. In general, separations of 3000 to 5000 meters and common velocities were used for initial conditions. Based on the relative geometry defined by these initial conditions, the algorithm operates on the current state vectors to compute acceleration commands along the axial and lateral body axes of the pursuing aircraft. These commands are limited to six g's laterally and two g's axially. During Lead Pursuit, the pursuer attempts to achieve an overtake velocity equal to 1.4 times the target's velocity in an

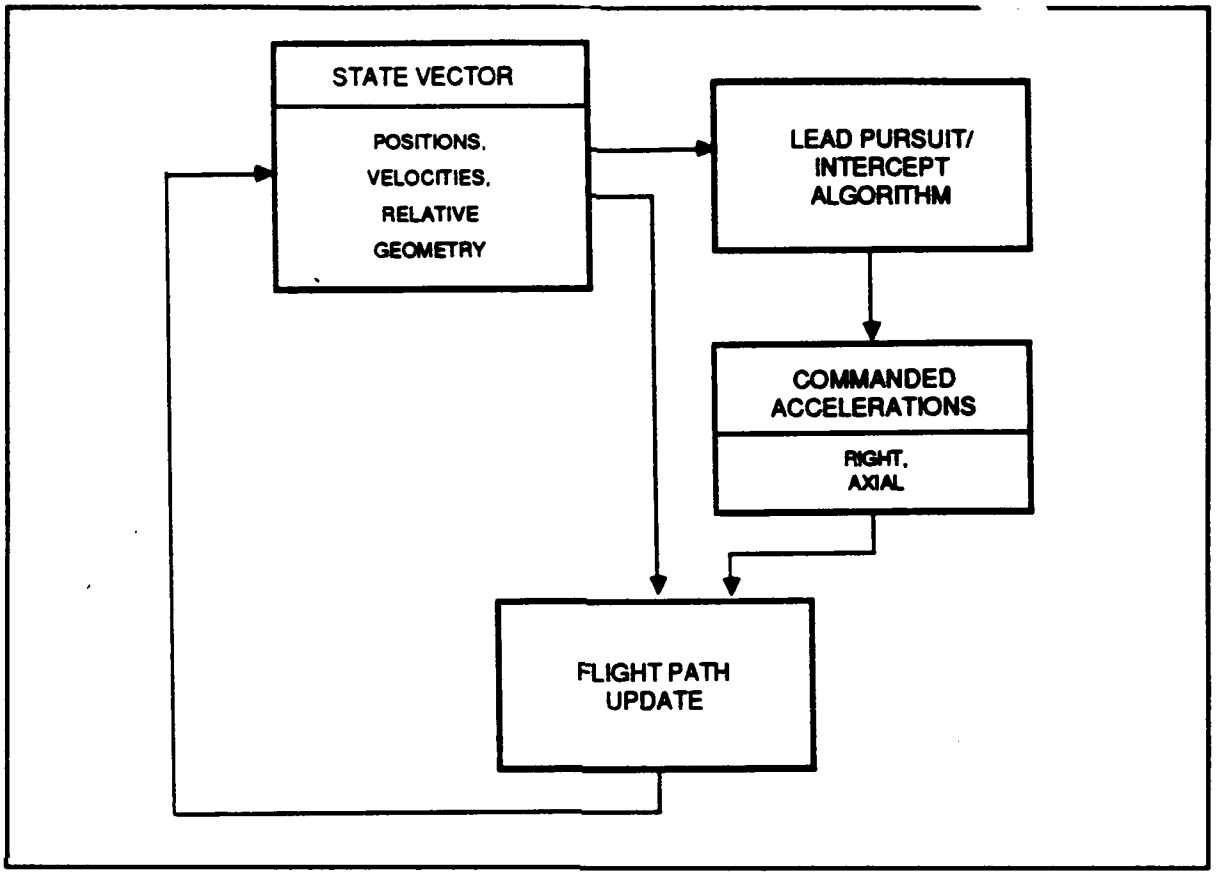


Figure 31. Flow of Control in the Lead Pursuit/Intercept Algorithm.

effort to close range. The acceleration commands, along with the current state vector are then sent to a flight path update routine which computes the new position, orientation, and airspeed of the aircraft and updates the state vectors. The target aircraft maintains a constant heading and airspeed unless acceleration commands are entered from the keyboard during run-time. At this point, the new positions of each aircraft are plotted, the input and output data are written to the network training file, and the cycle begins again. A set of complete flight paths for several starting conditions is shown as a two-dimensional plot in Figure 32.

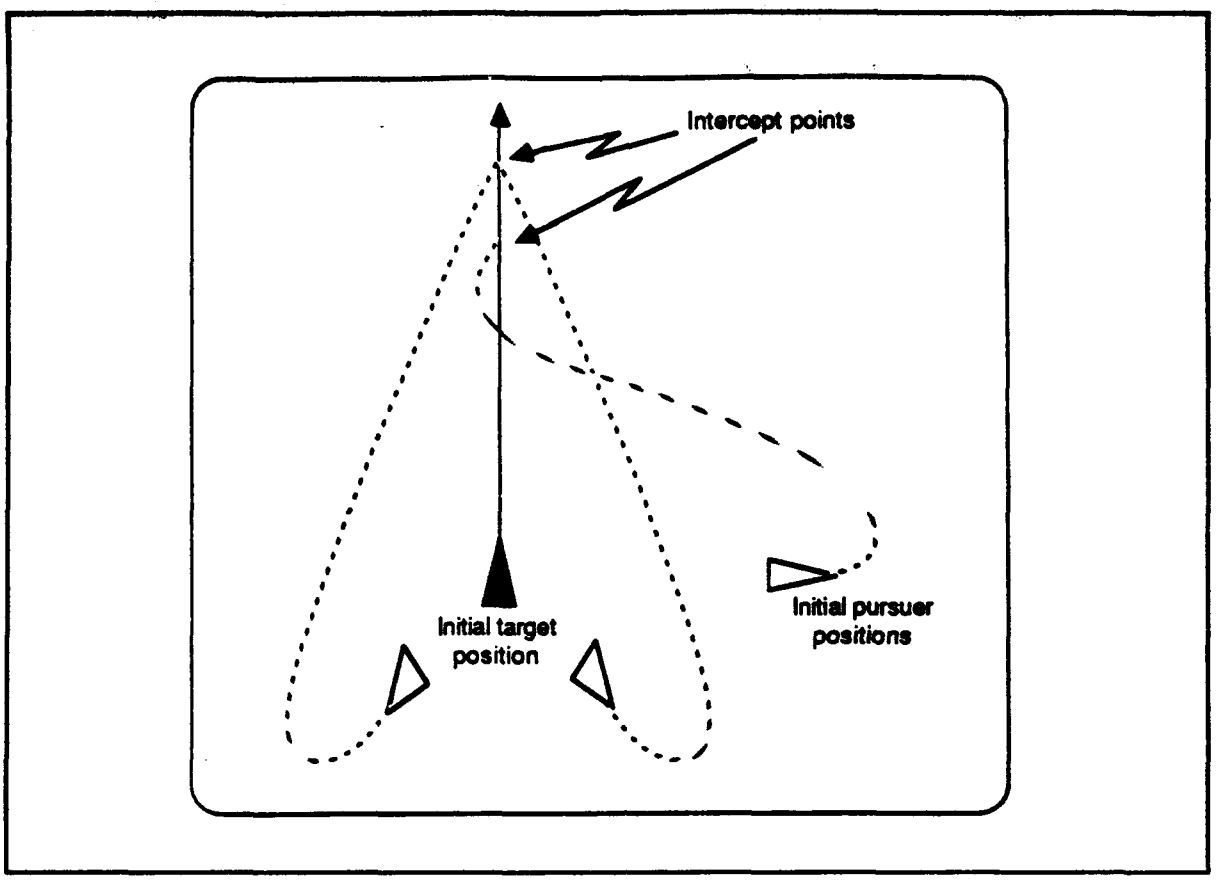


Figure 32. Sample Lead Pursuit/Intercept Profiles against a Nonmaneuvering Target.

Representation

To reproduce the behavior of the LPI algorithms using a neural network, a representation of the pertinent input and output parameters had to be developed. The relative geometry parameters become the inputs to the neural network, and the corresponding commanded accelerations are the outputs. As can be seen in Figure 33, the input layer of the network consists of processing elements which represent antenna-train-angle (ATA), target-aspect-angle (TAA), range between aircraft, closing velocity (V_c), and direction-independent velocity difference (ΔV). The final PE is a flag to indicate whether the Lead Pursuit or the Intercept phase is currently being used to generate output. On the output layer, the network simply generates the same output as the algorithm, namely, lateral acceleration and axial acceleration. ATA and TAA take on values from +180 to -180 degrees. Range varies from 0 to 5000 meters, V_c between ±300 m/s, and ΔV between ±90 m/s. Before presentation

to the network, all values are scaled between -1.0 and +1.0 to correspond to the proper PE activation range.

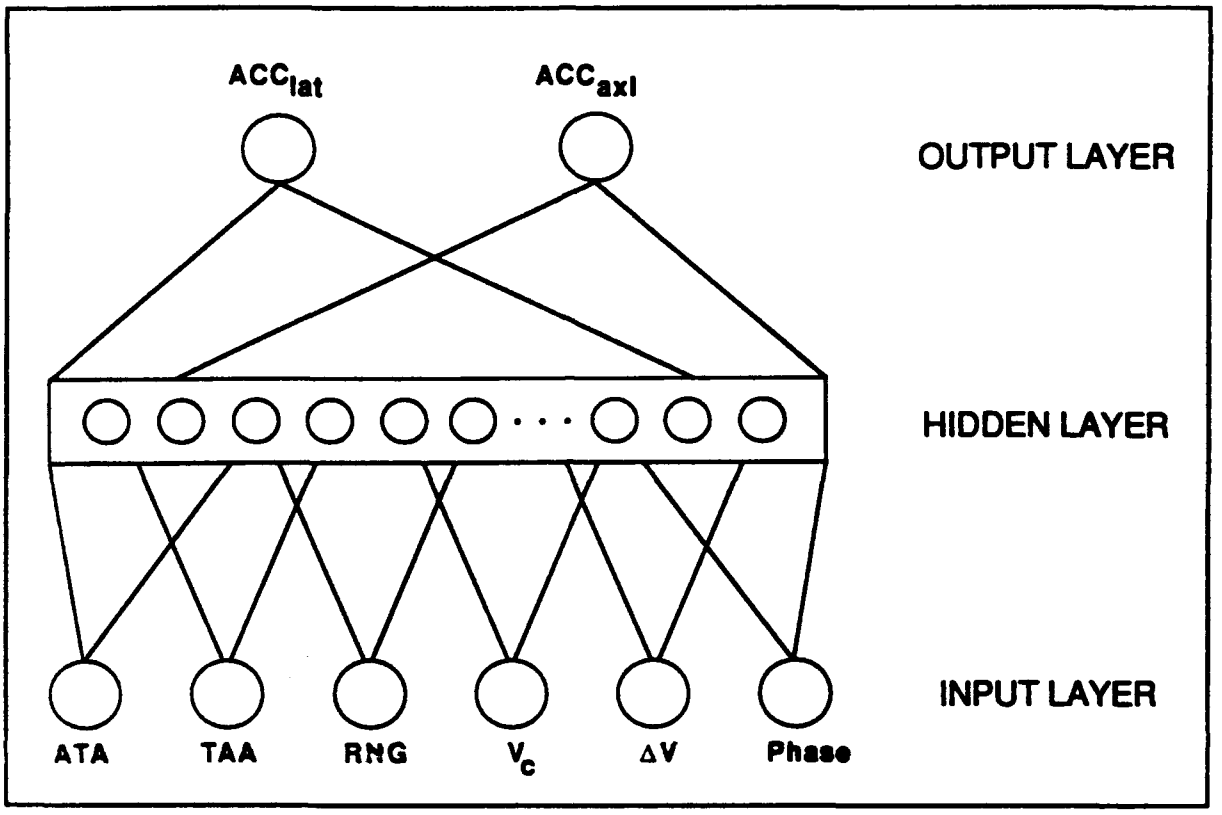


Figure 33. Network Structure of the Lead Pursuit/Intercept Demonstration System.

Generating Training Data

As has been discussed in the Neural Network Training Section and earlier in this section, training data must be selected in such a way as to constitute a uniformly distributed sample of input/output associations over the entire solution space. To approximate such a distribution, the network training file for the LPI demonstration was generated by selecting various initial conditions at regular intervals along each input variable's domain. For example, flight profiles were generated for starting angles of 45 to 315 degrees off the target's nose, incrementing by 45-degree intervals for each new profile. For each of these angles, the pursuer begins the flight profile with ATA values of 0, 90, 180, and 270 degrees relative to the target. Initial velocity and range values may be similarly sampled to produce a relatively distributed set of input conditions. The algorithm is then allowed

to iterate between 10 and 25 times for each set of initial conditions, creating a set of flight path fragments. Table 2 is a sample of the LPI neural network training set. The column headings in the table, reading from left to right, represent time, antenna train angle and target aspect angle relative to the pursuing aircraft, distance between the aircraft, closing velocity, absolute velocity magnitude difference, phase, and the outputs, lateral acceleration and axial acceleration.

Since only short fragments are used in the training file, some conditions must be generated where the initial range is small enough to ensure that the intercept phase occurs. In fact, since the intercept phase would naturally consist of far fewer data samples than the lead pursuit phase, extra samples of the intercept phase were presented to the network to balance out the preponderance of lead pursuit examples. In this way, the network learns how to accomplish the goals of each phase with equal effectiveness. When the LPI training file was constructed as described above, it contained approximately 1400 input/output associations similar to those shown in Table 2, which adequately spanned the entire solution space. Clearly, this is a very limited training set when it is considered that the entire range of possible starting conditions for a given flight profile is something on the order of 10^{13}, even if only whole number values are used.

Table 2. Sample Section of the Lead Pursuit/Intercept Training Data File

TIM	ATA	TAA	RNG	VC	DV	PHS	RAC	AAC
0	0.0	45.0	5000.0	-170.7	0.0	1	52.95	19.61
1	-27.1	45.5	4823.9	-176.5	19.6	1	7.41	19.61
2	-30.3	45.6	4640.4	-190.2	39.2	1	1.09	0.00
3	-30.7	45.7	4450.5	-189.6	39.2	1	0.04	0.00
4	-30.7	45.7	4260.9	-189.6	39.2	1	0.02	0.00
5	-30.7	45.7	4071.3	-189.6	39.2	1	0.02	0.00
6	-30.7	45.7	3881.7	-189.6	39.2	1	0.02	0.00
7	-30.7	45.7	3692.2	-189.6	39.2	1	0.02	0.00
8	-30.7	45.7	3502.6	-189.5	39.2	1	0.03	0.00
9	-30.7	45.7	3313.1	-189.5	39.2	1	0.03	0.00
10	-30.7	45.7	3123.6	-189.5	39.2	1	0.03	0.00
0	0.0	90.0	5000.0	-100.0	0.0	1	58.84	19.61
1	-29.9	90.8	4895.9	-102.3	19.6	1	32.80	19.61
2	-44.1	91.1	4794.5	-98.0	39.2	1	3.52	0.00
3	-45.6	91.1	4697.7	-95.5	39.2	1	0.05	0.00
4	-45.6	91.1	4602.2	-95.5	39.2	1	0.02	0.00
5	-45.6	91.1	4506.7	-95.5	39.2	1	0.02	0.00
6	-45.6	91.1	4411.2	-95.5	39.2	1	0.02	0.00
7	-45.6	91.1	4315.7	-95.5	39.2	1	0.02	0.00
8	-45.6	91.2	4220.3	-95.5	39.2	1	0.02	0.00
9	-45.6	91.2	4124.8	-95.4	39.2	1	0.02	0.00
10	-45.6	91.2	4029.4	-95.4	39.2	1	0.02	0.00
0	0.0	135.0	5000.0	-29.3	0.0	1	52.95	19.61
1	-27.1	135.5	4965.3	-35.1	19.6	1	6.10	19.61
2	-29.7	135.6	4922.9	-49.5	39.2	1	0.64	0.00
3	-30.0	135.6	4873.6	-49.1	39.2	1	0.01	0.00
4	-30.0	135.6	4824.5	-49.1	39.2	1	0.00	0.00
5	-30.0	135.6	4775.3	-49.1	39.2	1	0.00	0.00
6	-30.0	135.6	4726.2	-49.1	39.2	1	0.00	0.00
7	-30.0	135.6	4677.1	-49.1	39.2	1	0.00	0.00
8	-30.0	135.6	4627.9	-49.1	39.2	1	0.00	0.00
9	-30.0	135.7	4578.8	-49.1	39.2	1	0.00	0.00
10	-29.9	135.7	4529.7	-49.1	39.2	1	0.00	0.00
0	0.0	180.0	5000.0	0.0	0.0	1	-0.02	19.61
1	0.0	180.0	4990.2	-19.6	19.6	1	0.00	19.61
2	0.0	180.0	4960.8	-39.2	39.2	1	0.00	0.00
3	0.0	180.0	4921.5	-39.2	39.2	1	0.00	0.00

Training the Network

Training of the LPI network was accomplished through the use of the Neural Network Training System. The primary measure of effectiveness (MOE) during training was the minimization of MSE for the training set. The results of the networks were also tested against the performance of the algorithm for a broader comparison, as will be explained. A wide variety of network structures and training parameters were used to generate LPI networks which were scored using the above MOEs.

Though the size of the input and output layers was fixed by the data representation, both the number of hidden layers and the number of PEs in these layers was varied to determine the optimal structure. Many of the attempted internal structures were capable of forming a stable mapping, but the best overall structure consisted of a single hidden layer of 15 processing elements, with the output layer also receiving direct connections from the input layer. The resulting network consists of 23 PEs and 132 interconnections (excluding the 17 bias weights). Experiments with the training parameters yielded an optimal initial weight range value of ± 1.8, a learning rate (α) value of 0.15, and a momentum or smoothing term (β) of zero. A logistics activation function was used with a steepness value of 1.3. Training was carried out for 300,000 iterations, or approximately 200 passes through the training data set. The definition and resulting weight structure of the network created under this training process is shown in Appendix A. Using this set of weights, any set of six input values will result in a state change at the output layer which corresponds to the network's approximation for the appropriate acceleration commands to generate the intercept profile.

Comparing Algorithm and Network Performance

To validate the network's approximations of the LPI behavior, a flight profile, not just the specific acceleration values for the individual time steps, must be generated. By simultaneously plotting the flight profiles generated by the algorithm and the network, a visual examination provides the best analysis of relative performance. The time required to accomplish the intercept maneuver is also a good performance indicator, provided the network does not erroneously exceed the specified overtake velocity. To facilitate this comparison process, the Lead Pursuit/Intercept demonstration program was

created. A typical output display from the demonstration program is shown in Figure 34.

As can be seen in the figure, the display presents numerical position, orientation, and velocity data for one target aircraft and two pursuers. As the demonstration runs, the positions of the aircraft are also plotted, each in a different color, so that the evolving flight profiles can be compared. The target aircraft flies a due north, constant velocity profile, while the flight profiles of the pursuit aircraft are controlled by lateral and axial acceleration commands; one of the pursuit aircraft is controlled by the algorithm and the other receives its commands from the output layer of the neural network. The two pursuit aircraft begin the demonstration at exactly the same point, but the two controlling processes are completely independent, so the state vectors of the two aircraft may depart substantially with subsequent flight path updates. This can be seen in the plot of Figure 34, which is the result of an undertrained network (10,000 iterations). Figure 35 is a flow chart of the LPI demonstration, showing how control is passed among the various program modules.

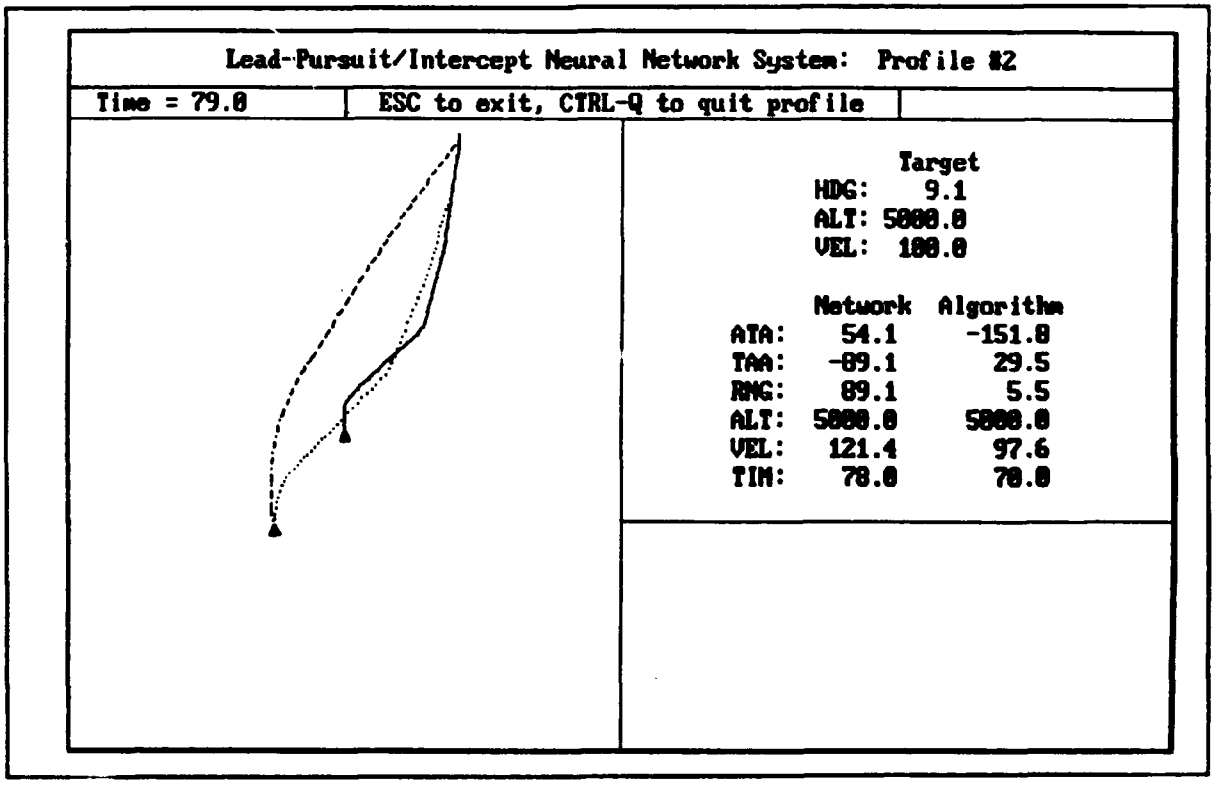


Figure 34. Sample Display Screen from the Lead Pursuit/Intercept Neural Network Demonstration.

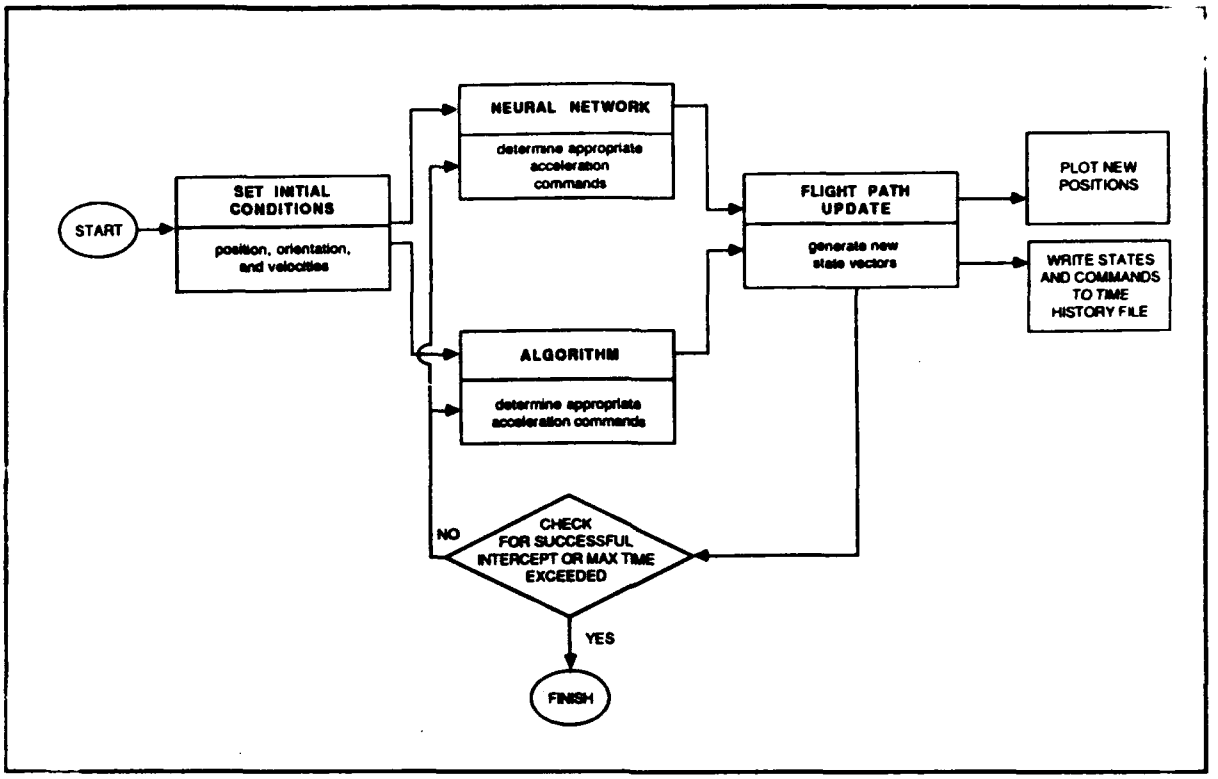


Figure 35. Flow of Control in the Lead Pursuit/Intercept Demonstration.

When using an optimally trained network to fly against a non-maneuvering target, the network displays a remarkably similar behavior to that of the algorithm, Figure 36. In most cases, the two pursuit profiles lie directly on top of one another from the initial position until intercept. There is often a slight difference in the number of time cycles required for the pursuit aircraft to accomplish the intercept maneuver, with the network being slightly slower than the algorithm. In every case, however, the neural network is able to achieve the intercept goal and does so in much the same manner as the algorithm. The precise simulation of the algorithm by the network proved to be a somewhat surprising result, particularly since the network was trained on such a sparse sampling of the algorithm's solution space. Further experimentation with the system turned up even more interesting results.

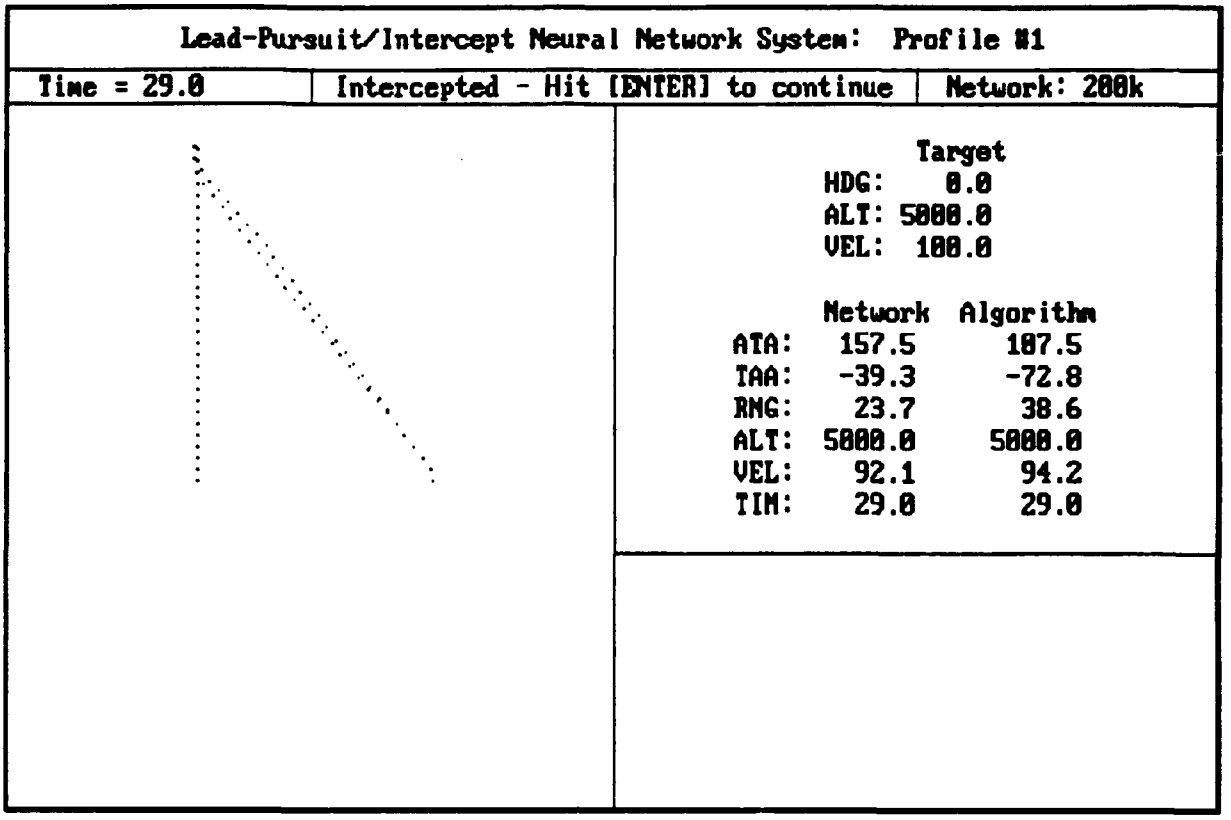


Figure 36. Lead Pursuit/Intercept Demonstration - Nonmaneuvering Target.

The LPI demonstration software allows the user to inject acceleration commands into the target aircraft during run time, either via the arrow keys of the keyboard or by "flying" the target aircraft with a joystick or mouse. This creates a maneuvering target which can fly into a variety of unusual relative geometries not included in the network's training set. Even though the network was not trained under these conditions, it will still respond to any input conditions, generating its closest approximation to output based on the mapping that resulted from exposure to the training set. For example, if the user increases the target velocity well above the velocity ranges used in training, the network responds correctly by increasing its own velocity until the proper overtake velocity is reached. A corresponding reduction in velocity is produced when the target aircraft is slowed down by the user. In these cases, it can be seen that the network has developed a meta-level concept to code for overtake velocity based only on the relative velocity information contained in the training set, Figure 37.

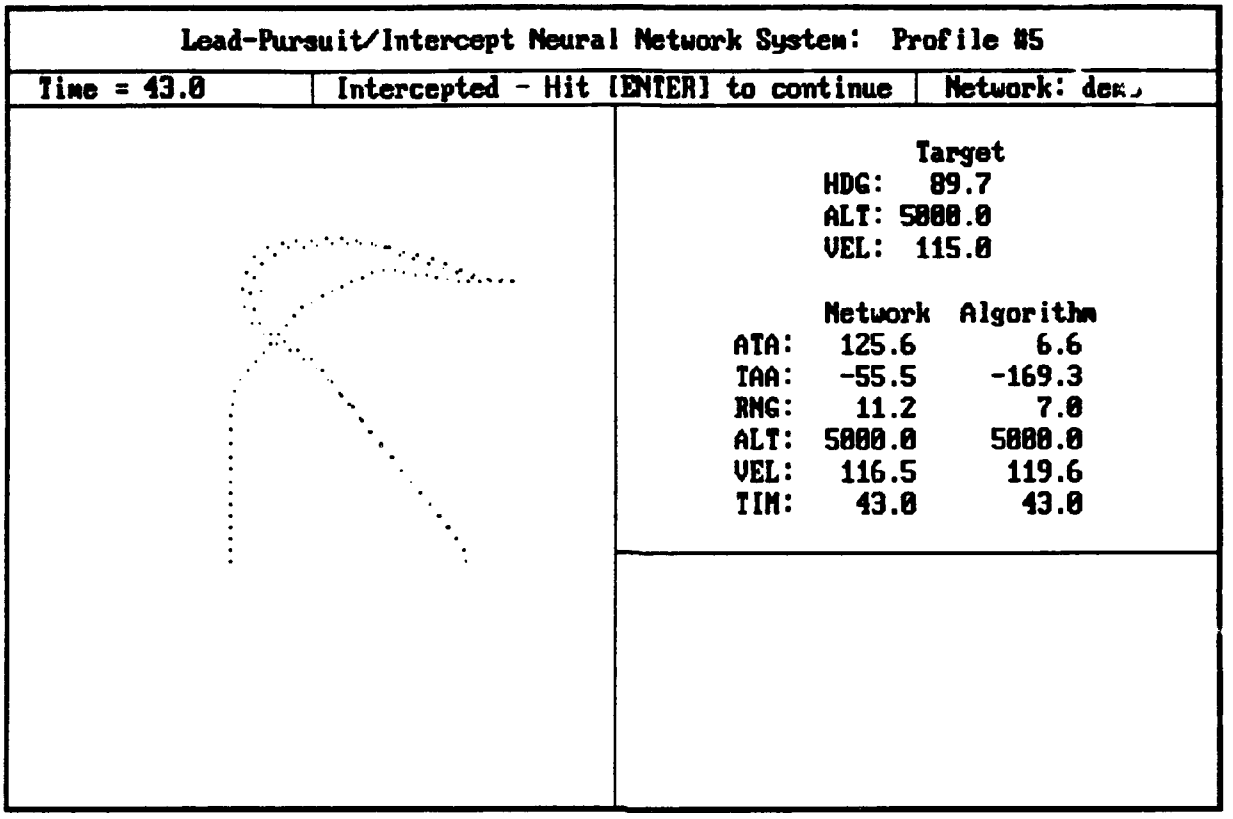


Figure 37. Lead Pursuit/Intercept Demonstration - Maneuvering Target.

Not only can the network respond to extreme examples of typical training cases, it can also produce successful responses to completely novel geometry conditions. In fact, under certain conditions when the target aircraft is maneuvering, the network will produce a much more direct flight path to intercept than the algorithm, Figure 38. This is possible due to a number of factors. First, note that the algorithm does not always perform what appears to be the optimal flight path to accomplish the LPI goal. Certain geometries cause the algorithm's equations to break down temporarily, generating errant acceleration commands which perturb the overall flight path. Since these cases are relatively isolated, the equations were not altered, and conditions that were known to cause a failure of the algorithm were not used in the generation of training data. Hence, the neural network knows nothing about this poor performance of the algorithms since it was only exposed to "good" data. Another factor is that the network has arrived at a broad generalization to the LPI mapping based on its exposure to the limited but well-distributed training set. When presented with novel conditions, this generalization provides an approximate response which is an interpolation of

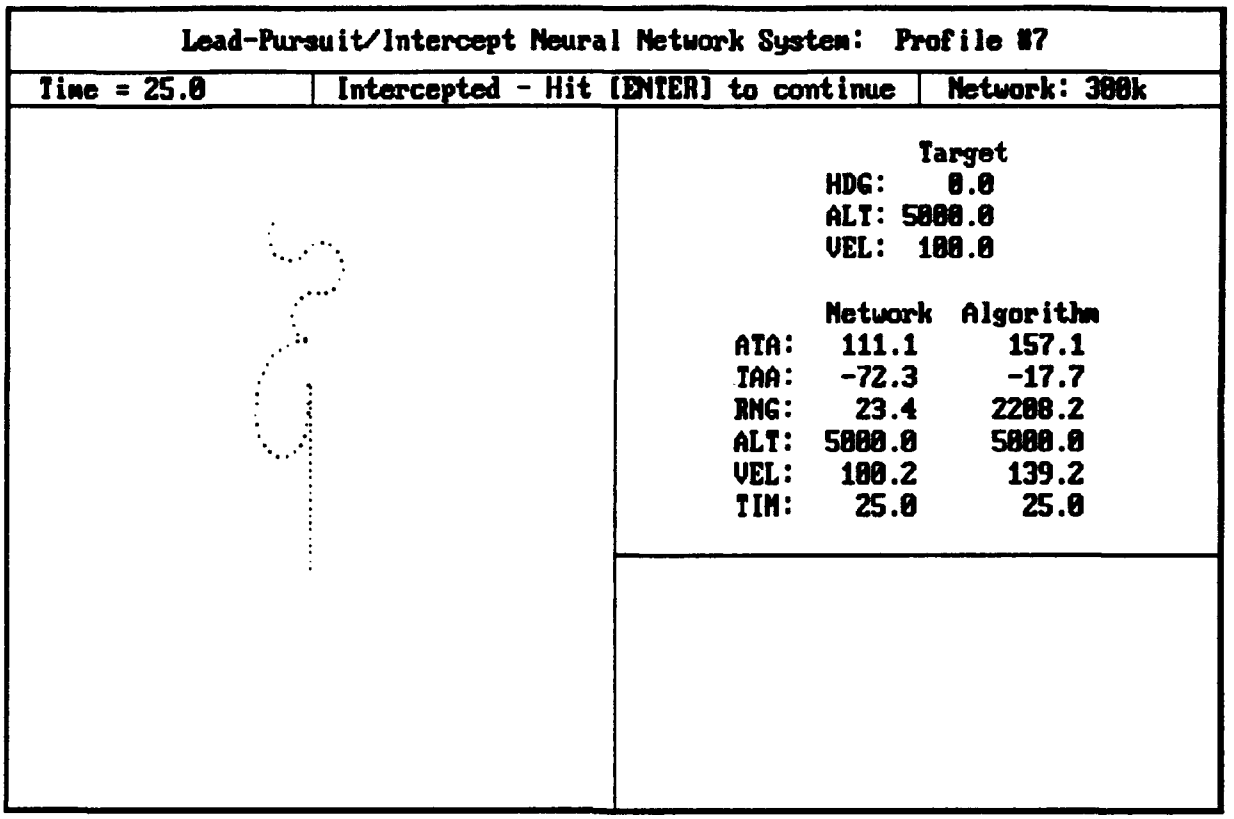


Figure 38. Lead Pursuit/Intercept Demonstration - Direct Path.

conditions it has been trained upon. In this way the neural network is able to outperform its teacher because of the algorithm's inflexibility and the network's ability to generalize a reasonable response.

Scaling Up to the ACM Expert System

The Lead Pursuit/Intercept demonstration system has met or exceeded all of its design goals: providing insight into how neural networks behave, verifying that an ANS representation fits well with the kinds of data required to model the ACM regime, and resulting in a wealth of validated software to be used in the development of the ACM Expert System. There is a direct mapping between the various modules of the LPI software and the requirements of the ACM Expert System. For instance, the same MBPN Neural Network Training System is used to train ACM networks using Simulator for Air-to-Air Combat (SAAC) performance data as is used to produce the LPI networks. Also, the ANS support tools developed and used in the previous experiments are carried over

to the later phases of the program with no modifications. Instead of using flight data from the LPI algorithms, real pilot performance data forms the basis of the ACM Expert System's training set, and the network structure and representation described below replaces the LPI network. To provide the system with accurate aerodynamics for flight path update, the Blue Max II flight profile generator replaces the simple update software used in the LPI system. The bulk of the control code to interface with the ANZA Plus and the User Interface Subroutine Library is carried over with only minor modifications.

ARTIFICIAL NEURAL SYSTEM FOR THE REPRESENTATION AND COLLECTION OF ACM DECISION-MAKING EXPERTISE

Introduction to ARCADE

Based on the success of the Lead Pursuit/Intercept Demonstration system, a prototype neural network-based ACM Expert System, ARCADE, was developed. ARCADE is an acronym for Artificial neural system for the Representation and Collection of Air combat maneuvering Decision making Expertise. The first step in creating ARCADE was to define an appropriate representation scheme for the neural network's input and output vectors. This representation was based on the appropriateness and availability of training data, and on the requirement to drive a specific aircraft profile generation model. After a satisfactory representation had been developed, a data transfer process was set up to gather ACM performance data from the SAAC to be used for neural network training. The data was then preprocessed into the proper format. Various network structures were constructed and trained using the chosen representation. The performance of these networks was validated using a variety of objective and subjective means. Successful neural networks have been incorporated into ARCADE, an interactive, computer-based simulation which allows two aircraft to fly air combat against one another. Usually, one aircraft is controlled by a human user while the other is controlled by an ARCADE network; however, it is possible for both of the aircraft in ARCADE to be controlled by neural networks. This feature, in effect, allows for fly-offs to be held between two ARCADE network-controlled aircraft. The flight profiles of each aircraft are presented via the two-dimension, three-view display of ARCADE, Figure 39.

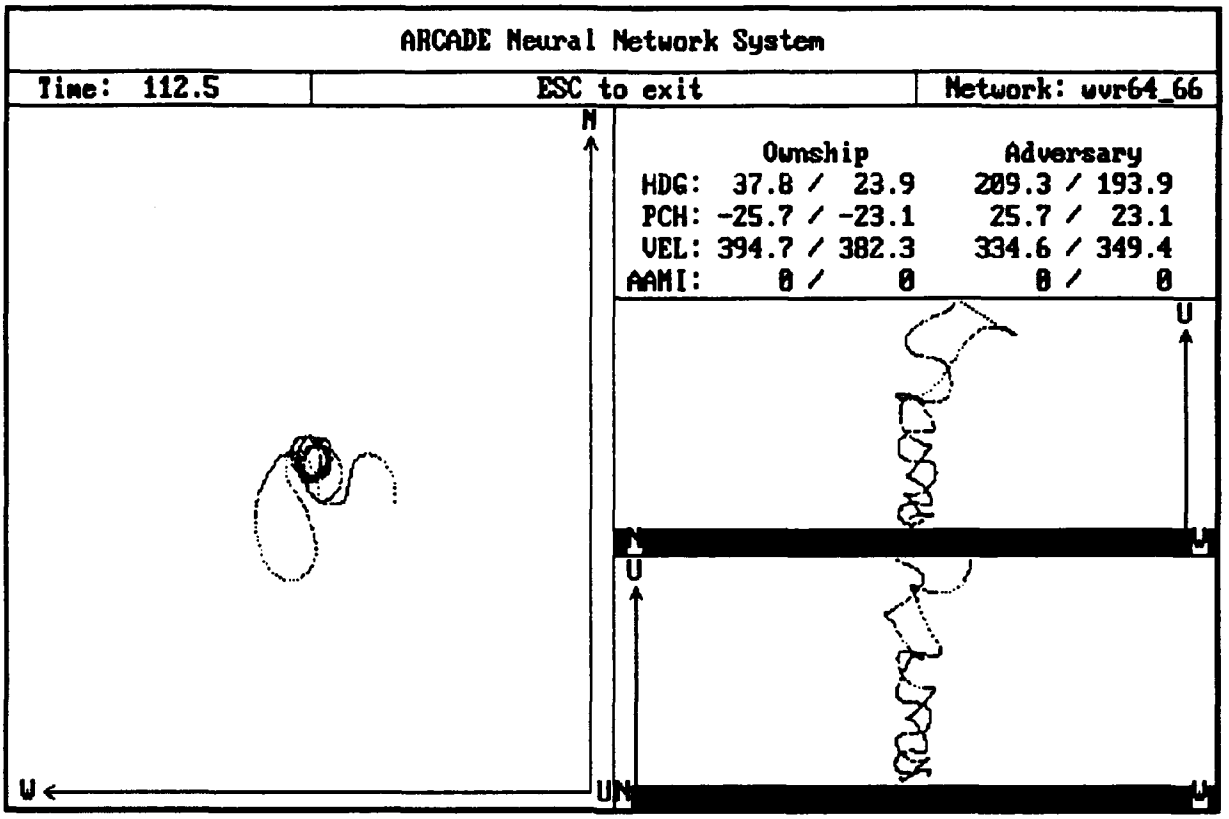


Figure 39. ARCADE Display.

The internal architecture and the flow of control of ARCADE can be seen in Figures 40 and 41, respectively. The initial conditions (aircraft heading, pitch, velocity, etc.) of the user and network aircraft are read from a user-defined file. See Figure 42 for an example of the aircraft initial conditions file. During ARCADE execution, the user controls the ownship by adjusting the aircraft's commanded heading, pitch and velocity via the computer keyboard keypad. The heading is adjusted with the left/right arrow keys and the pitch angle is modified with the up/down arrow keys. The velocity is increased/decreased with the plus/minus keys. The adversary aircraft is controlled via an ARCADE neural network. Given the current tactical situation, the ARCADE network provides the adversary aircraft with heading, pitch and velocity commands. The flight commands for both the user and network aircraft are then passed to the flight path generator, Blue Max II, where the states of the aircraft (x, y, z, heading, pitch, velocity, etc) are updated to reflect the current heading, pitch and velocity commands. The

position of the two aircraft are presented by ARCADE via a two-dimension, three-view display, Figure 39.

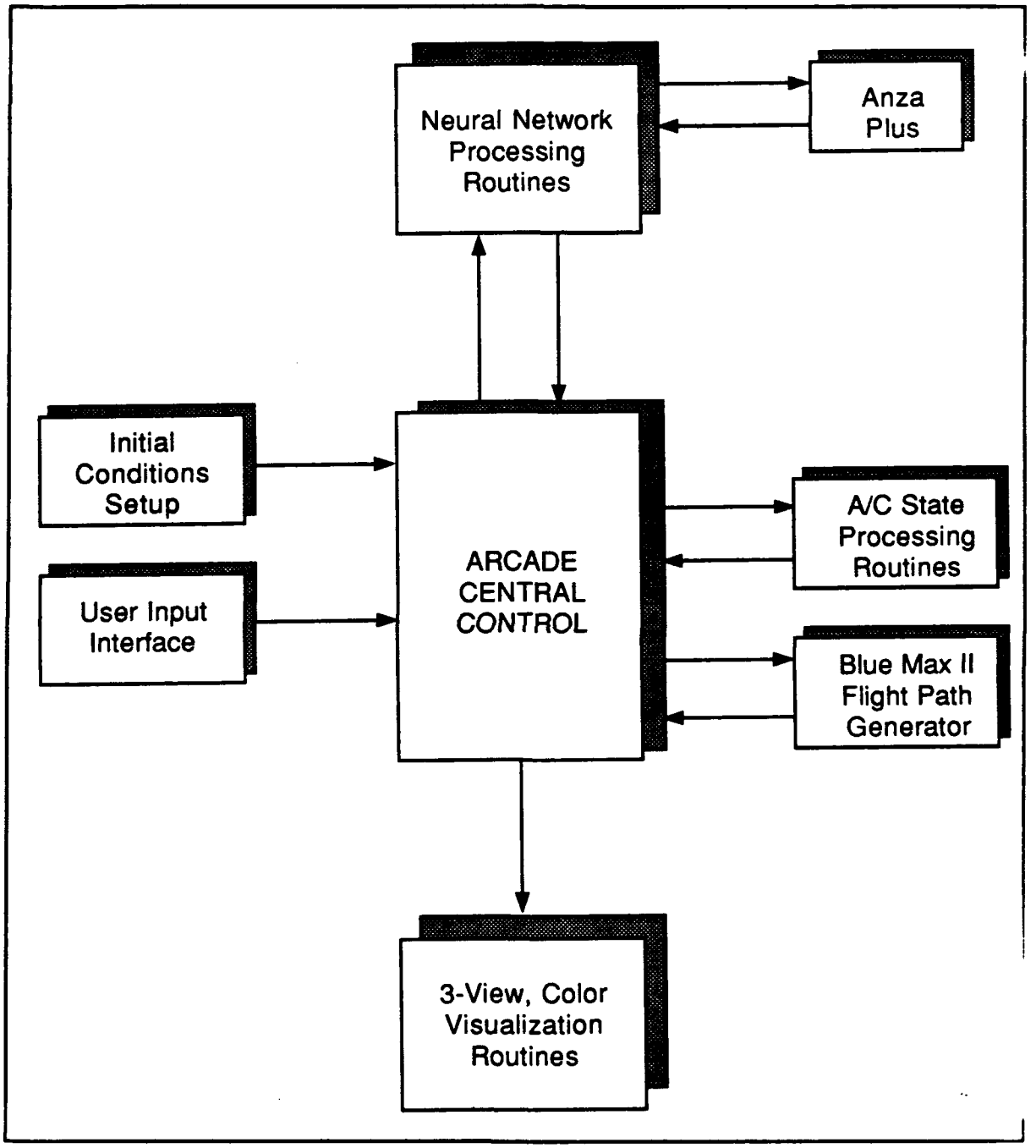


Figure 40. Internal Architecture of ARCADE.

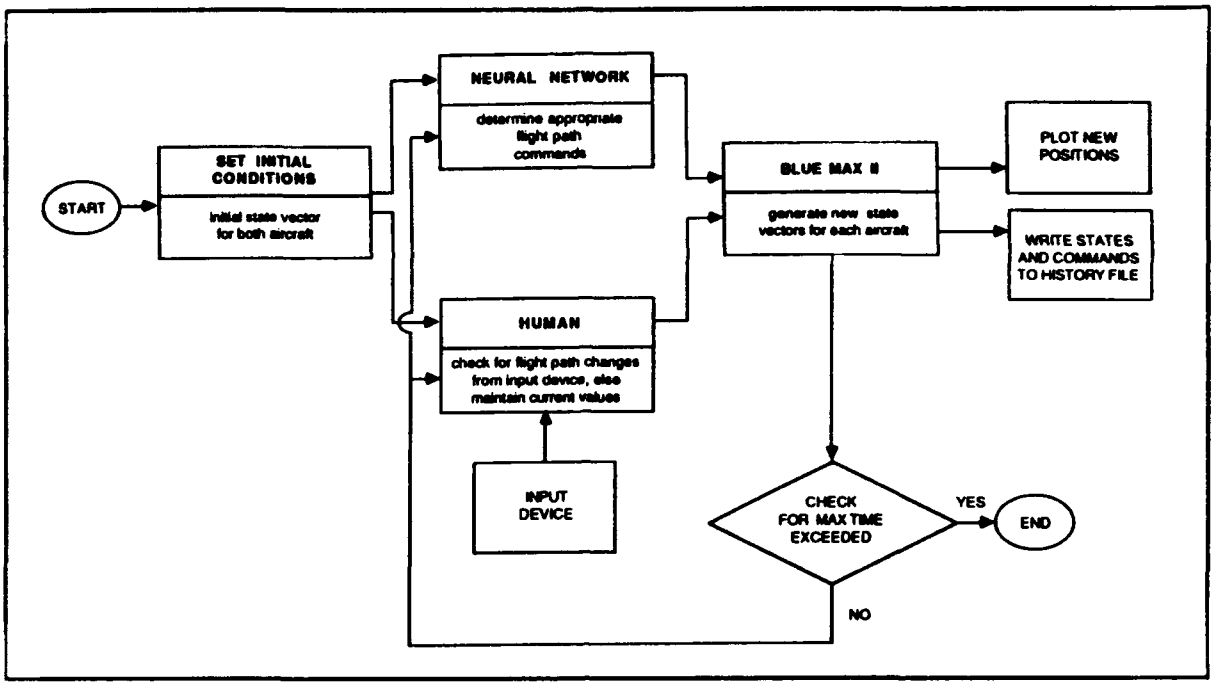


Figure 41. Flow of Control of ARCADE.

```
  0.5              SYSTEM DATA: UPDATE RATE (SEC)
  0.0     0.0 15000.0  USR AC: X (FT), Y (FT), Z (FT)
  0.0     0.0   800.0          HDNG (DEG), PTCH (DEG), VEL (FT/SEC)
  0.0 10000.0     0.0  ANS AC: A-O-N (DEG), RANGE (FT), DELTA ALT (FT)
180.0     0.0   800.0          HDNG (DEG), PTCH (DEG), VEL (FT/SEC)
```

Figure 42. ARCADE Initial Conditions File.

The flight path generator is a primary component of ARCADE. It takes the desired maneuver commands from the ARCADE network, as well as the user, and generates trajectories consistent with aircraft flight characteristics. Blue Max II was selected as the flight path generator to be used by ARCADE.

Since it is the function of the flight path generator to update the position and orientation of the aircraft per the commands provided by the ARCADE networks, the profiles generated by Blue Max II were compared with the profiles of the SAAC engagements. A Blue Max II validation utility, BM2Valid, was developed to validate the Blue Max II flight path generator against the training data collected from the SAAC. The maneuver commands generated by the

SAAC were passed to Blue Max II and the profile generated was plotted against the original SAAC flight profile. Ideally, the profile generated by Blue Max II would be identical to the SAAC engagement profile. It was discovered that Blue Max II, while performing reasonably well in many cases, was not able to perform all of the maneuvers present in the SAAC engagement data. For example, Blue Max II is unable to duplicate SAAC maneuvers where large heading changes occur with the aircraft at extreme pitch angles, as when an aircraft passes through the vertical. In addition, controlling the adversary aircraft via heading, pitch and velocity commands does not allow the ARCADE network to control the aircraft's roll angle. The roll computed by Blue Max II may or may not be the optimal roll angle. It is believed that these anomalies do degrade the performance of the ARCADE networks, but the extent of the degradation is unknown.

ARCADE Neural Network Architecture

The development of a network structure for ARCADE was approached by first determining a minimal set of tactical input data that might be required by a combat pilot when making ACM decisions in a one-versus-one engagement. Similarly, the components of the output response were chosen to meet a minimal set of aircraft flight control commands to direct aircraft maneuvering. The time dependency of the dynamic tactical situation data is also represented in the input vector.

An overview of the internal architecture of the ARCADE neural network is illustrated in Figure 43. The following describes how the various parameters of the tactical situation/maneuver response vectors are represented at the input and output layers of the ARCADE neural network. When describing the current tactical situation and maneuver response, the aircraft controlled by the neural network is called the ANS aircraft and the other aircraft, usually controlled by a human user, is referred to as the adversary aircraft. All relative variables are computed relative to the ANS aircraft.

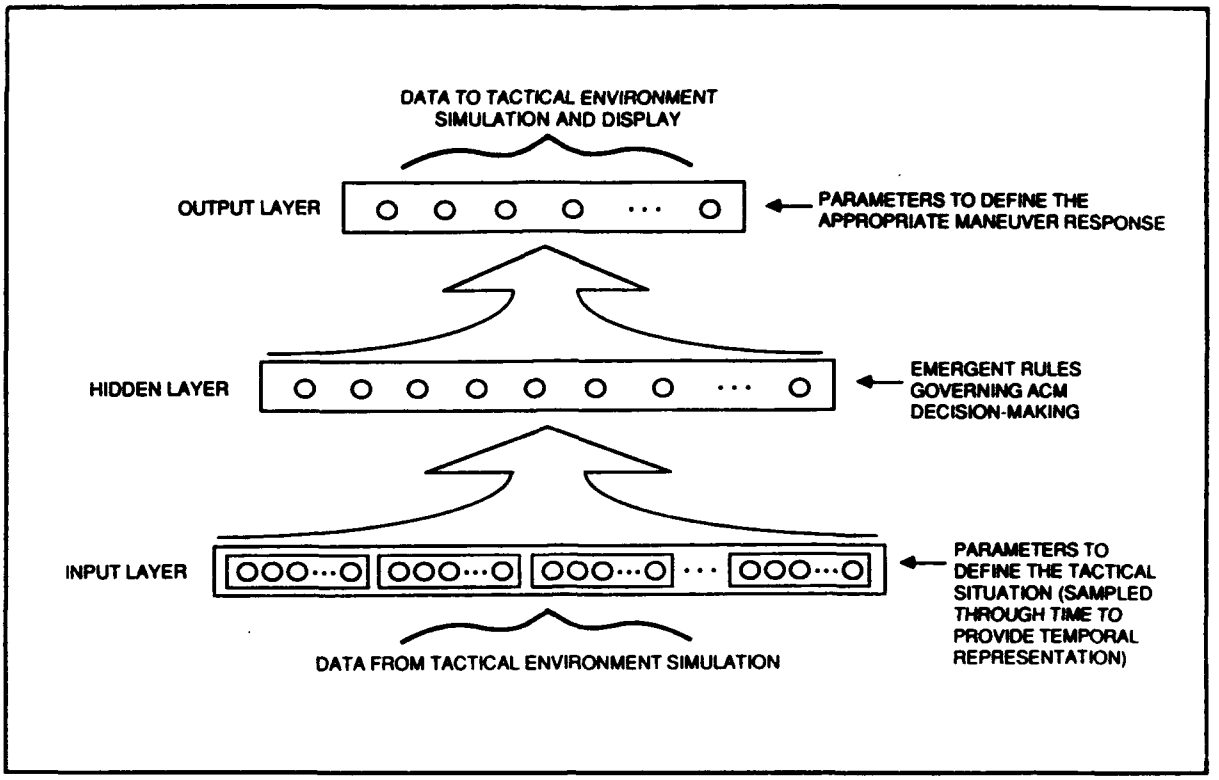


Figure 43. ARCADE ANS Internal Architecture.

Input Layer

In the ARCADE neural network, the input vector contains parameters which define the current tactical geometry in terms of slant range, azimuth angle to adversary, angle-off-tail (AOT) in the horizontal plane, closing velocity (V_c), elevation angle to adversary, ANS aircraft's altitude, ANS aircraft's angle-of-attack (AOA), ANS and adversary aircraft's velocity, ANS and adversary aircraft's pitch angle, ANS and adversary aircraft's roll angle, ANS and adversary aircraft's turn rate, and ANS aircraft's G factor. See Table 3 for the ARCADE network input parameters and their operating ranges. Certainly, there are other dynamic parameters which may be included, but this set meets the development criteria set forth previously and seems to capture enough information about ANS aircraft status and the relative location of the adversary aircraft to provide a basis for reasonable and realistic ACM performance.

Table 3. ARCADE Network Input Parameters and Operating Range

Slant Range	: 0 to 13,000 feet
Azimuth Angle	: +/- 180 degrees
Angle-Off-Tail (AOT)	: +/- 180 degrees
Closing Velocity (V_c)	: +/- 1,500 ft/sec
Altitude	: 0 to 20,000 feet
Elevation Angle	: +/- 90 degrees
Angle-of-Attack (AOA)	: -10 to 25 degrees
Velocity	: 100 to 1,000 ft/s
Adversary's Velocity	: 100 to 1,000 ft/s
Pitch	: +/- 90 degrees
Adversary's Pitch	: +/- 90 degrees
Roll	: +/- 180 degrees
Adversary's Roll	: +/- 180 degrees
Turn Rate	: 0 to 45 deg/s
Adversary's Turn Rate	: 0 to 45 deg/s
G Factor	: -1 to 9 g's

The set of network input parameters used in ARCADE are listed in Table 3. They are defined as follows. The direct line of sight (LOS) angle from the ANS aircraft's longitudinal axis to the adversary aircraft's position is defined by the azimuth angle. The distance between the two aircraft, along the LOS, is defined by slant range. The Angle-Off-Tail (AOT) provides information about the orientation of the adversary aircraft relative to the line of sight in the horizontal plane. V_c indicates the rate of closure of the two aircraft (positive if the aircraft are closing range). The angle from the horizontal plane in which the ANS aircraft resides to the adversary aircraft is defined by the elevation angle. The absolute altitude of the ANS aircraft is also included. AOA is the angle that the ANS aircraft's wing makes with

the direction of airflow. Velocity parameters provide information about the absolute, ground velocity of each aircraft. Roll and pitch angles provide aircraft orientation information. The turn rates and G factors indicate how quickly each pilot is pulling his aircraft's nose across the sky. Each of the above parameters was chosen for the input vector because it represents a piece of crucial, top-level, dynamic data relevant to the evolving ACM environment. The basic geometry of a typical ACM tactical situation and some of the input vector parameters are illustrated in Figures 44 and 45.

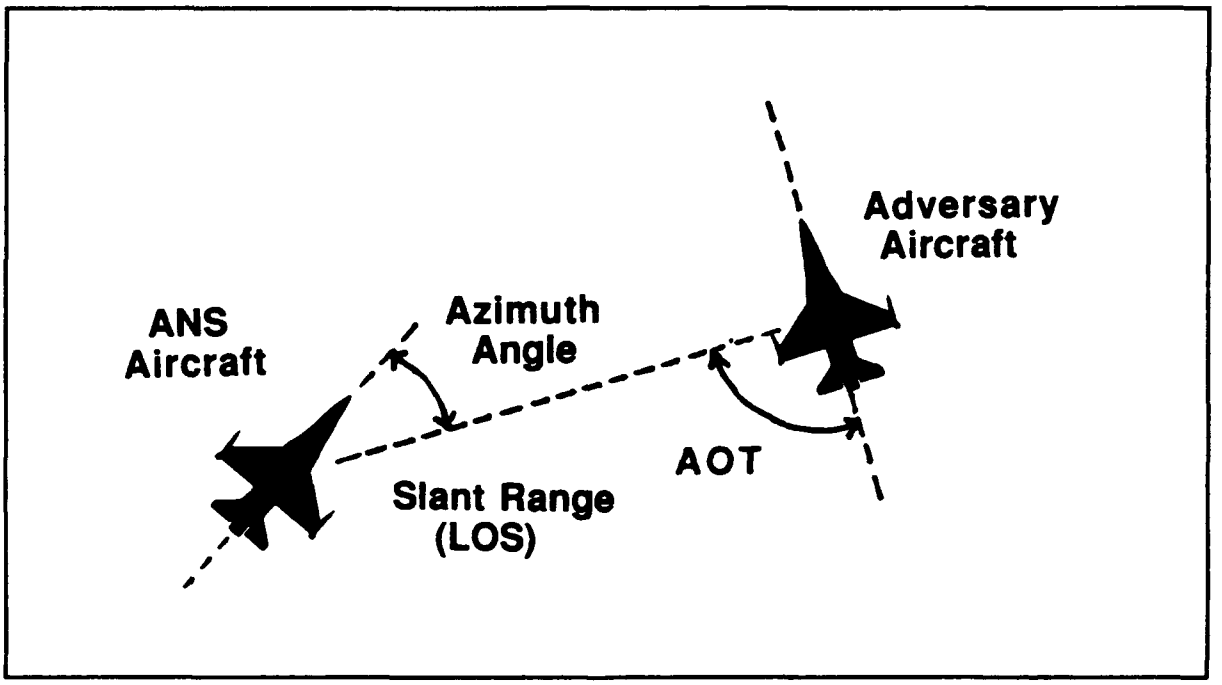


Figure 44. Typical Air Combat Geometry - Top View.

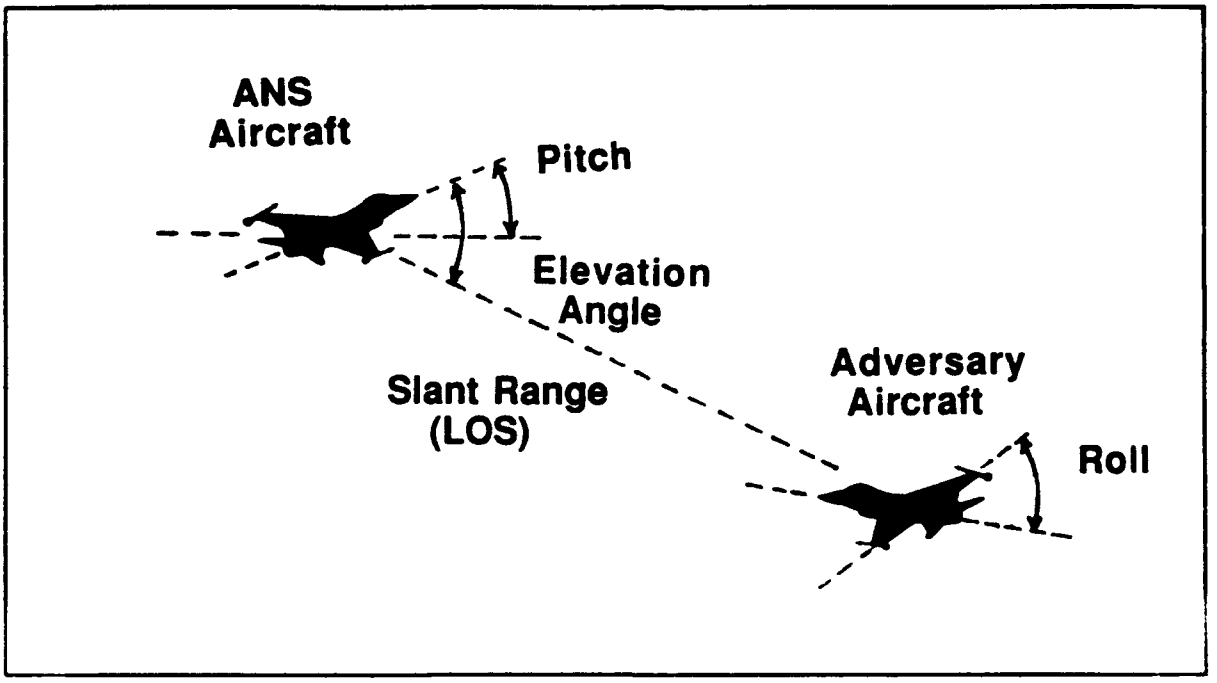


Figure 45. Typical Air Combat Geometry - Side View.

Another important aspect of the input feature vector is the element of time. A pilot makes his maneuvering decisions based not only on a "snapshot" of the current tactical situation, but also on how the situation has been evolving over time. In the ARCADE neural network, the input feature vector contains multiple copies of the basic parameter set described above. Each copy represents a different point in time over the last few seconds. The resulting input layer thus consists of one set of input parameters for the present situation, another for the situation one half second ago, and additional sets for two, and five seconds in the past. This adds up to a total of 64 input PEs (4 sets of 16) as depicted in Figure 46. As with the basic set of input parameters, this is a preliminary design which may be changed if different time periods or sample times seem to produce better ACM performance. The current version of the ARCADE neural network operates at two cycles per second. Every half second, the system is provided with new tactical input data from the last five seconds to produce an associated maneuver response.

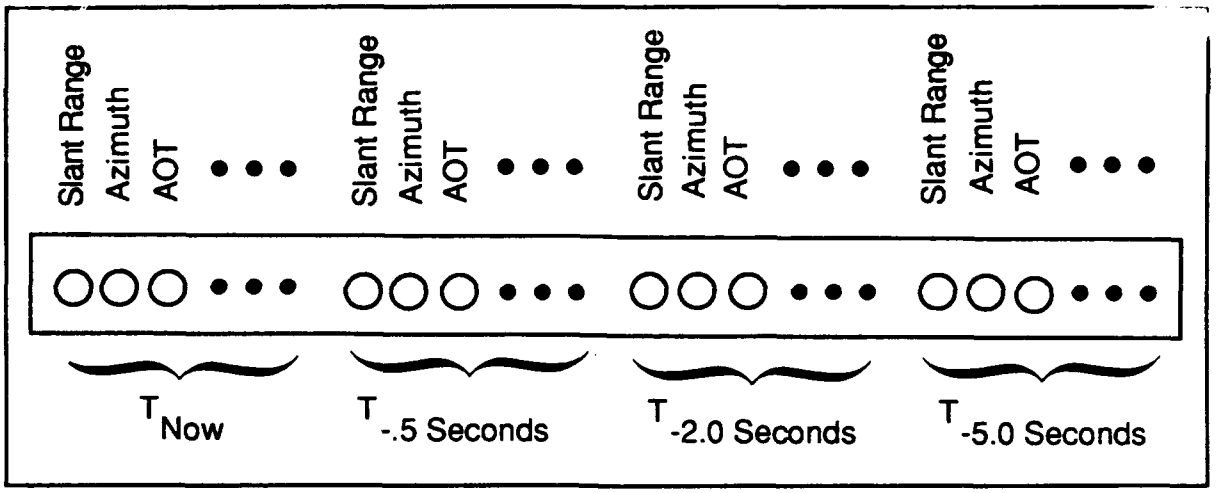


Figure 46. ARCADE Neural Network Input Layer.

Output Layer

At the output layer of the ARCADE neural network, the system has been designed to produce output parameters compatible with the flight commands required by the Blue Max II aerodynamic model. The output processing elements of the ARCADE network therefore represent three flight path commands for continuous control: a desired change in heading, pitch, and velocity. These output variables are referred to as **delta command** variables because they represent the difference between the current and desired value for each flight control parameter. The neural network could be trained to produce another set of flight control parameters, such as roll rate, turn rate and velocity, if necessary. The current set of output parameters and their operating ranges are listed in Table 4.

Table 4. ARCADE Network Output Parameters and Operating Ranges.

Delta Heading	: +/- 60 degrees
Delta Pitch	: -50 to 30 degrees
Delta Velocity	: -120 to 80 ft/s

Unlike the input vector, the output PEs need not contain multiple representations over time. Each time the neural network is provided with a set of tactical parameters for the last few seconds, it will provide the aircraft flight model with the set of flight path delta commands to control the aircraft. It is necessary, however, to use a time window on the SAAC output data stream to determine the desired flight path commands relative to the time now. In addition to the time sampling window for the input layer, Figure 47 shows how such a time window is used to read the output data parameters. While the input values are based, in part, on the previous conditions, the output values are determined by looking ahead from the current maneuver state to the state as it will appear a few seconds later. The heading, pitch and velocity values on the SAAC data tape at that future time are used to compute the current desired output values for the network. As with the input window parameters, the exact size of the output data window has been varied experimentally to ascertain an optimal time projection of 1.5 seconds.

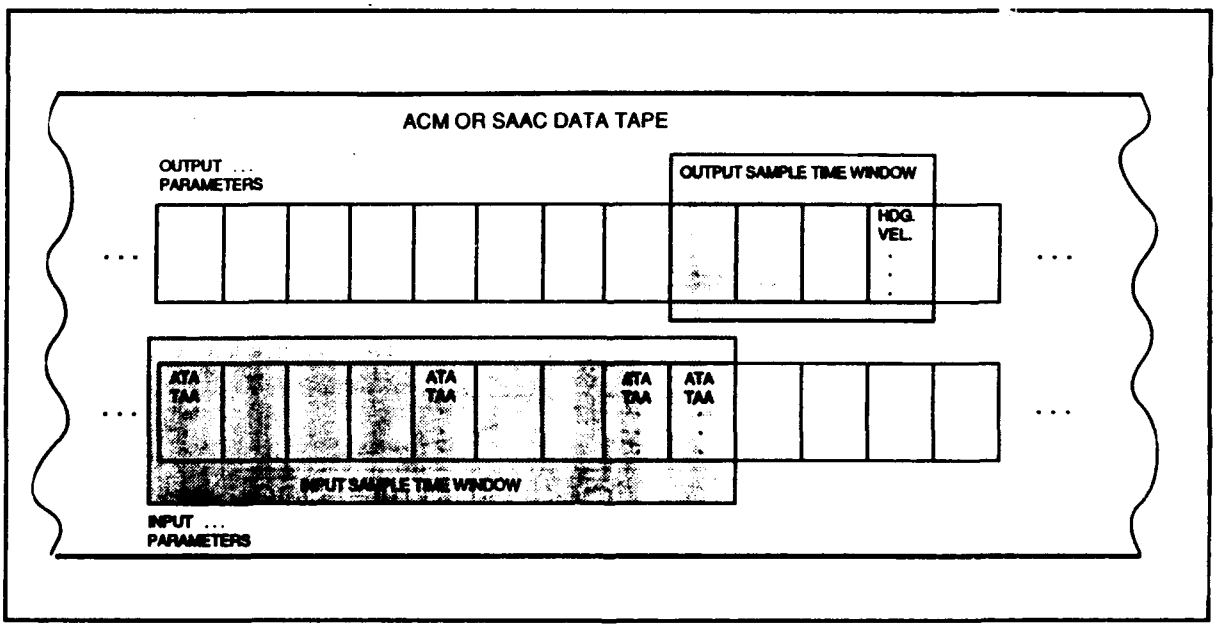


Figure 47. Data Windows for Reading Information from the SAAC ACM Data Tapes.

ARCADE Network Training Process

The ARCADE network training process consists of the collection and preprocessing of training data, the training of ARCADE networks, and the performance evaluation of these networks.

Training Data Collection and Preprocessing

Initially, it was thought that the Air Combat Maneuvering Instrumentation (ACMI) would provide the best source of data, but after some consideration, it was determined that data from the SAAC at Luke AFB in Arizona would yield the best overall results. The nature of the SAAC exercises and data storage format allows a more direct specification of the type of data selected for training. There is a certain amount of performance evaluation data available with each scenario, and there is generally a wealth of data available to choose from.

The simulator engagement conditions which were used to generate the training data are as follows. Subjects flew one-versus-one exercises using identical aircraft, F-16 versus F-16. The aircraft were equipped with unlimited AIM-9L missiles and gun rounds. Each pilot was instructed to continue the fight until there is a kill or the controller terminates the engagement. The maximum engagement time is generally three minutes in duration. The aircraft begin the engagement within visual range (WVR), either side-by-side with one nautical mile of separation or head-to-head with two nautical miles of separation. Both aircraft begin the engagement at 15,000 feet traveling at 500 nautical miles per hour. These are reasonable conditions for creating a complete and consistent set of ACM network training data.

The "ACM Expert System Analyst Manual" contains samples of the SAAC engagements used to train the ARCADE neural networks. The engagements are displayed via a two-dimension, three-view display similar to that used by ARCADE. The large display is a top dowr view while the two smaller displays are side views. Aircraft kills are depicted with an "X" for AIM-9L hits and an "O" for gun hits. Note that because the engagements are between pilots of similar skill levels flying identical aircraft, each aircraft spends most of

the engagement in a relatively neutral position with few of the shots fired resulting in an adversary aircraft kill.

Once the ACM engagement data had been collected, it was converted into the temporal tactical situation/maneuver response neural network input/output vectors described above. Two network training segments were created from each SAAC engagement; one from aircraft number one's perspective and a second from aircraft number two's perspective. The network training data generated from the SAAC engagements were then grouped into several training and testing sets. These training and testing sets are listed in Table 5. "SAAC_12.TRN" is comprised of training segments selected from the original 12 SAAC engagements collected at the program's inception. Later in the program, it was deemed necessary to collect an additional 20 engagements from the SAAC. "SAAC_32.TRN" is comprised of training segments selected from the expanded set of 32 SAAC engagements. Not all of the SAAC engagements were used for network training. "SAAC_32.TST" contains SAAC engagements that were intentionally withheld from the network training sets for network evaluation purposes. All of the engagements collected from the SAAC were used for network training or testing. No attempt was made to preprocess, classify or grade the engagements collected from the SAAC.

Table 5. SAAC Training and Testing Data Files

ARCADE Training File Name: SAAC_12.TRN
SAAC engagements included in training file:
040106 (2), 040115 (1&2), 040211 (1&2), 040313 (1&2), 420305 (1&2), 420306 (1&2), 420309 (1&2), 500205 (1&2), 500210 (1&2), 500305 (1&2), 510205 (1&2), 510214 (1&2)
ARCADE Training File Name: SAAC_32.TRN
SAAC engagements included in training file:
040106 (2), 040115 (1&2), 040211 (1&2), 040313 (1&2), 420305 (1&2), 420306 (1&2), 420309 (1&2), 500205 (1&2), 500210 (1&2), 500305 (1&2), 510205 (1&2), 510214 (1&2), 070307 (1&2), 080209 (1&2), 080410 (1&2), 080507 (1&2), 110417 (1&2), 120115 (1&2), 120317 (1&2), 140208 (1&2), 200211 (1&2), 151505 (1&2), 262512 (1&2), 300315 (1&2), 320206 (1&2)
ARCADE Testing File Name: SAAC_32.TST
SAAC engagements included in testing file:
040106 (1), 070206 (1&2), 110306 (1&2), 200308 (1&2), 250106 (1&2), 260115 (1&2), 300512 (1&2)

ARCADE Network Training

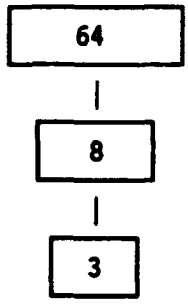
With the ARCADE training and testing data generated, it was possible to begin training ARCADE neural networks. ARCADE network training was accomplished with the use of the Neural Network Training System. Several neural networks were trained while experimentally varying the size of the hidden layer, the number of hidden layers, the initial network weights, the weights' update process, the learning rates, the network training data and the duration of network training. Appendix D contains the network weights and constants' file name, network structure, training parameter definitions, and network training and testing statistics for several of the networks developed for ARCADE.

Using ARCADE Network Experiment #1 found in Appendix D as an example, the ARCADE network development process proceeded as follows. Based on earlier experimentation, practical experience and intuition, the network structure and training parameters were defined. A network consisting of one hidden layer with eight PE's was selected. The weights were initialized randomly using a random number seed of 12345 and a maximum range of +/- 0.2. A logistics activation function with a slope of 1.0 was selected. The learning rates for the output/hidden layers were set to 0.1/0.6, respectively. The largest, most robust training set available, SAAC_32.TRN, was selected as the network training set. Batching was selected as the weight update process. The batch size was set equal to the average number of tactical situation/maneuver response pairs per SAAC engagement, 250.

While ARCADE networks were being trained using the Neural Network Training System, the Mean Squared Error (MSE) and Mean Absolute Error (MAE) were generated. The goal of the Neural Network Training System was to minimize MSE. Several times during ARCADE Network Experiment #1, the constants and weights of the network were saved and MSE/MAE recorded, see Table 6. Network training was continued until MSE and MAE began to flatten out to MSE/MAE values of 0.085/0.443, respectively. The length of training was approximately fifteen million iterations, which translates to several hours of network training.

Table 6. ARCADE Network Experiment #1 - Network Definition and MSE/MAE.

Network Structure



Network Parameters

Random Weights Seed: 12345

Initial Weights Range: +/- 0.2

Activation Function: Logistic
Slope: 1.0

Learning Rates: 0.1, 0.6

Batch Size: 250

Training Set: SAAC_32.TRN

Testing Set: SAAC_32.TST

Network Name	Iterations	Training MSE/MAE
WVR_15	250,000	0.155 / 0.488
WVR_16	500,000	0.136 / 0.448
WVR_18	750,000	0.126 / 0.428
WVR_19	1,250,000	0.115 / 0.404
WVR_20	1,500,000	0.112 / 0.393
WVR_21	2,250,000	0.104 / 0.378
WVR_22	2,500,000	0.103 / 0.376
WVR_23	3,250,000	0.101 / 0.371
WVR_24	3,750,000	0.099 / 0.368
WVR_26	5,500,000	0.097 / 0.362
WVR_27	10,500,000	0.090 / 0.334
WVR_28	12,500,000	0.088 / 0.342
WVR_29	14,500,000	0.086 / 0.337
WVR_30	15,500,000	0.085 / 0.334

The rest of the ARCADE Network Experiments found in Appendix D were conducted in a similar manner. The network structure, training parameters and/or training set were experimentally modified and the training process repeated.

ARCADE Network Performance Evaluation

The performance of the ARCADE networks was evaluated by employing both objective and subjective means.

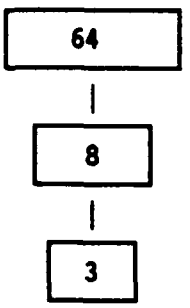
Objective Network Performance Evaluation

Objective evaluation of the ARCADE networks is comprised of Mean Squared Error (MSE) and Mean Absolute Error (MAE), statistics which indicate how well the network maps temporal tactical situations to maneuver responses. Continuing with experiment #1 of Appendix D, the previously saved ARCADE networks were tested to see how well they were able to arrive at a generalized solution when exposed to SAAC engagements that were withheld from network training. The test set, SAAC_32.TST, is comprised of SAAC engagements that were intentionally withheld from the network training set. Using the Neural Network Training System, each saved network was tested on the engagements in the test set and the MSE and MAE values generated were recorded, see Table 7. These statistics are an excellent indication of the network's ability to arrive at a generalized solution within the tactical situation domain of the SAAC engagements.

Since the training set is not a complete set of the tactical situations expected to be encountered in a dynamic, run-time environment, it is desirable to have an ARCADE network capable of generalizing the training data to arrive at reasonable responses to unique tactical situations rather than "lock-in" on only the tactical situations found in the training set. Likewise, as a network is trained on the tactical situation/maneuver response examples, the network will reach a point where additional training will decrease that network's ability to arrive at a generalized solution. For this reason, it is reasonable to expect that the best ARCADE networks would be the networks that minimize the test MSE and MAE rather than the training MSE and MAE. In

Table 7. ARCADE Network Experiment #1 - Test MSE/MAE.

Network Structure



Network Parameters

Random Weights Seed: 12345

Initial Weights Range: +/- 0.2

Activation Function: Logistic
Slope: 1.0

Learning Rates: 0.1, 0.6

Batch Size: 250

Training Set: SAAC_32.TRN

Testing Set: SAAC_32.TST

Network Name	Iterations	Training MSE/MAE	Testing MSE/MAE
WVR_15	250,000	0.155 / 0.488	0.133 / 0.451
WVR_16	500,000	0.136 / 0.448	0.122 / 0.423
WVR_18	750,000	0.126 / 0.428	0.115 / 0.406
WVR_19	1,250,000	0.115 / 0.404	0.108 / 0.387
WVR_20	1,500,000	0.112 / 0.393	0.106 / 0.382
WVR_21	2,250,000	0.104 / 0.378	0.102 / 0.373
WVR_22	2,500,000	0.103 / 0.376	0.101 / 0.371
WVR_23	3,250,000	0.101 / 0.371	0.099 / 0.368
WVR_24	3,750,000	0.099 / 0.368	0.098 / 0.365
WVR_26	5,500,000	0.097 / 0.362	0.094 / 0.356
WVR_27	10,500,000	0.090 / 0.334	0.088 / 0.337
WVR_28	12,500,000	0.088 / 0.342	0.087 / 0.336
WVR_29	14,500,000	0.086 / 0.337	0.086 / 0.330
WVR_30	15,500,000	0.085 / 0.334	0.086 / 0.331

the case of experiment #1 of Appendix D, the network saved after 14.5 million training iterations, WVR_29, had the lowest test MAE value, 0.330. See Figure 48.

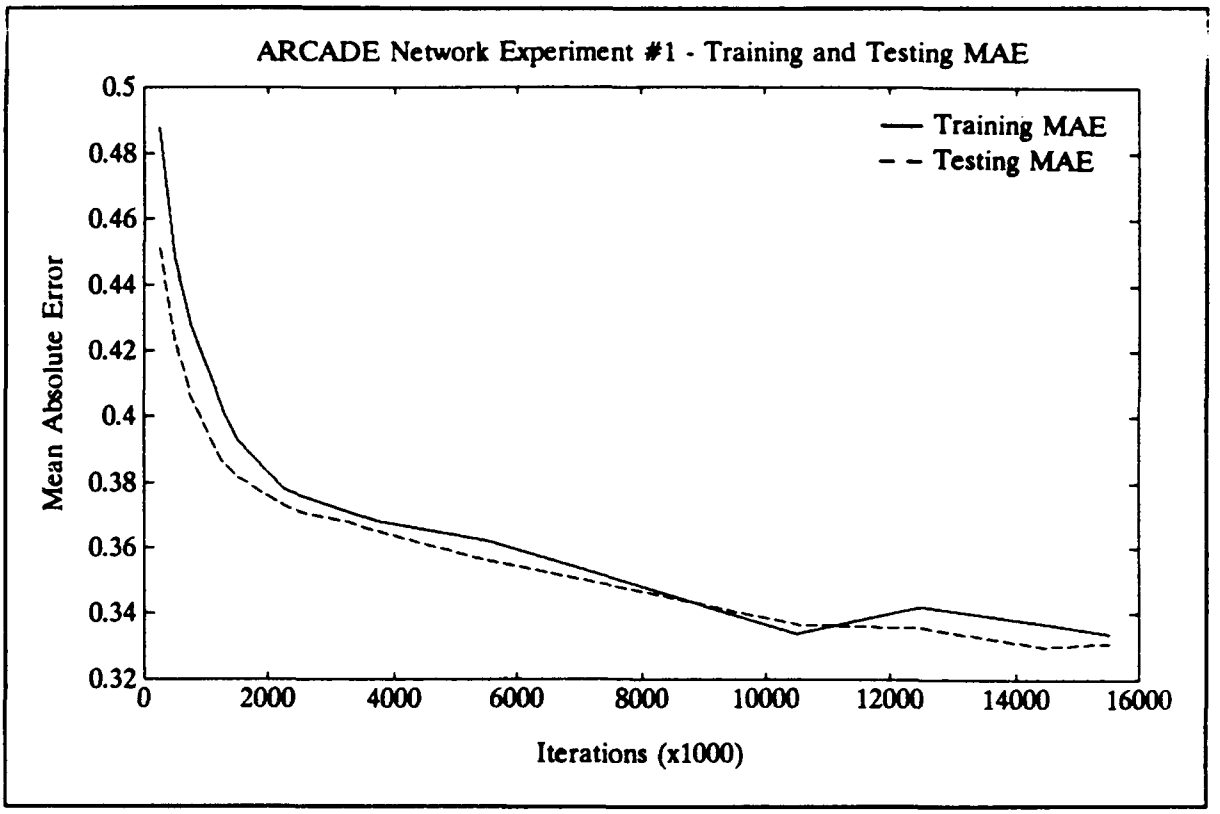


Figure 48. ARCADE Network Experiment #1 - Training and Test MAE.

Subjective Network Performance Analysis

Subjective evaluation of the ARCADE networks takes place in the run-time environment of ARCADE. Using ARCADE, a human user maneuvers an aircraft controlled by the computer keyboard against an adversary aircraft controlled by one of the previously trained ARCADE neural networks.

In order to subjectively evaluate the performance of an ARCADE network, it is crucial to be familiar with the SAAC engagements that the network is designed to replicate. It should be the goal of the ARCADE user to present the ARCADE network with tactical situations which are typical of those present in the SAAC engagements of which the network was trained. It is not, however, necessary for the user to try to create tactical situations which are

identical to those found in the training set. The ARCADE networks should be able to generalize to unique tactical situations within the domain of the SAAC engagements on which they were trained.

Figure 49 shows an engagement between a human user, with many hours of ARCADE experience, against an adversary aircraft controlled by ARCADE network WVR_29 -- the network of ARCADE Network Experiment #1 which had the lowest test MAE. The engagement begins with each aircraft flying 500 knots, head-to-head at 15,000 feet, with two nautical miles of separation. Each aircraft turned east upon passing one another and proceeded into what resembles diving-rolling scissors. Both the user and adversary aircraft terminate their descent at approximately 500 feet to avoid impact with the ground. The entire engagement is contained within the cylindrical area between 15,000 feet and the ground, two nautical miles in diameter.

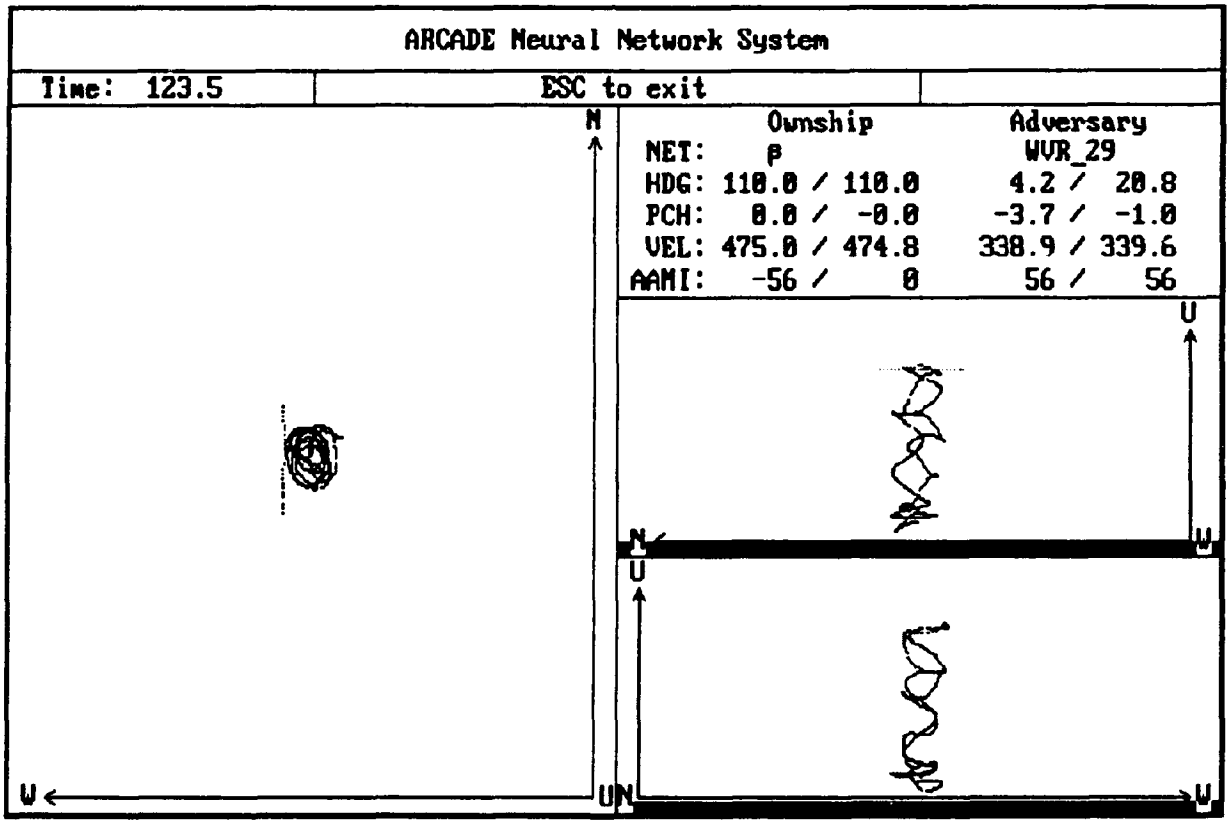


Figure 49. ARCADE Engagement - User Aircraft versus Adversary WVR_29.

Given the difficulty in flying effective ACM from a computer keyboard, it is effective to allow two aircraft controlled by ARCADE networks to engage one another. Figure 50 shows an engagement between two ARCADE network controlled aircraft, each aircraft controlled by the ARCADE network WVR_29. This engagement is an example of two equally matched adversaries engaged in air combat, as was true with a large portion of the SAAC engagements. Notice that this engagement also exhibits the characteristics present in the SAAC engagements. It is a well-contained engagement, each aircraft exchanging an altitude advantage while descending toward the ground.

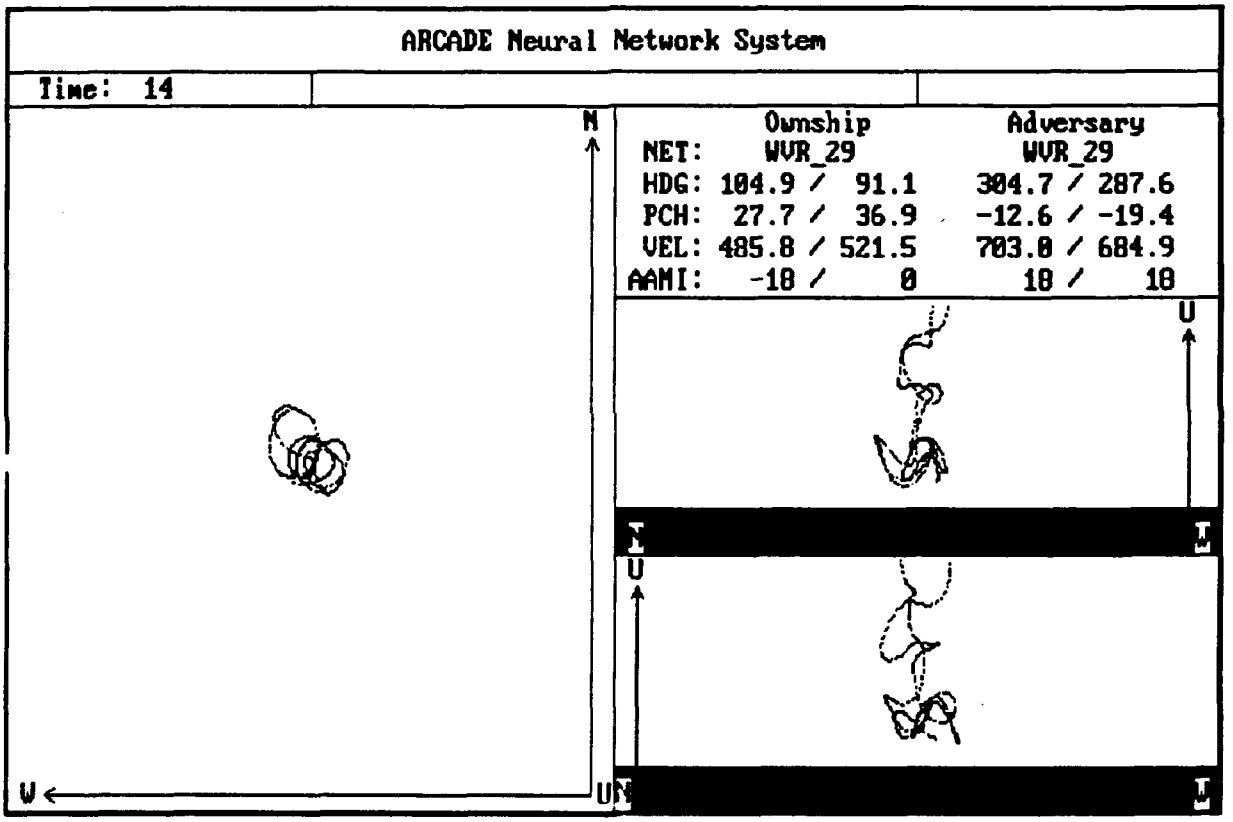


Figure 50. ARCADE Engagement - Adversary WVR_29 versus Adversary WVR_29.

Conclusions of Network Performance

An ARCADE network is believed to possess the ability to generalize unique solutions within the problem domain when the mean absolute error (MAE) of the test set is approximately equal to the MAE of the training set. The objective

network performance evaluation of ARCADE networks indicate that several networks have been created which possess the ability to arrive at generalized solutions to unique tactical situation within the domain of the SAAC engagements. The objective network performance evaluation does not, however, give any indication of the ARCADE network's ability to generalize to a reasonable solution for tactical situations which are outside of the domain of the SAAC data.

The ultimate evaluation of ARCADE networks is its performance in the run-time environment of ARCADE. While many ARCADE networks produce flight profiles similar to SAAC engagement profiles, poor network performance has been observed in ARCADE. It is presumed that the poor performance occurs when the network is presented with tactical situations outside the domain of the SAAC engagements. That is, the user and adversary aircraft have maneuvered themselves into a tactical situation that is outside of the domain of the SAAC engagements on which the ARCADE network adversary was trained. For example, while the ARCADE networks are capable of performing diving-rolling scissors against an opponent aircraft, the networks do not provide optimal maneuver commands when presented with a non-maneuvering target. Engagements with non-maneuvering targets do not appear in the SAAC engagement training set.

In conclusion, both the objective and subjective evaluation of network performance indicate that an ARCADE network controlled adversary is capable of performing reasonable and realistic air combat against a human pilot. However, the operational domain of the ARCADE network adversary appears to be smaller than the desired set of tactical situations in which the network provides optimal responses. It is believed that the operational domain of the ARCADE networks could be expanded to the point of optimal performance under all conceivable tactical conditions with the methodical collection of an expanded set of SAAC engagements.

GENERAL FINDINGS AND CONCLUSIONS

Neural Network Structure

With respect to the structural definition of a particular neural network, it can be said that once the input and output layers are defined as a result

of the chosen input/output representation, the construction of the hidden layer or layers is largely the result of educated guesses and trial-and-error. The best approach seems to be to start small and work up to progressively larger networks, keeping in mind that more PEs mean more interconnections, and therefore, slightly longer training times. Furthermore, too many PEs in the hidden layer will interfere with the network's ability to form a general solution to the mapping, and can cause it to perform preferentially well on the training data while failing at unique situations. There is usually an optimal number of PEs which provides the best network performance for the test set while avoiding the loss of generalization. The actual number of hidden layers should be kept to one unless additional layers show a marked improvement in either mean absolute error, mean squared error or general production performance.

Spanning the Solution Space and Generalization

A fundamental conclusion of the neural network research conducted for this program is that the ability to correctly reproduce expert behavior from exemplar data is critically dependent on how the input/output association is represented and how the training data is selected. The concept of spanning the solution space is a crucial element in determining the final performance of the network under the full range of operating conditions. The process in which input/output data is sampled from the total solution space will directly affect the final mapping at which the system arrives. This mapping will either allow the system to successfully generalize its solutions from known to unique conditions, or cause erratic behavior in all but a few input cases. The experiments previously described, especially the Lead Pursuit/Intercept demonstration, clearly illustrate the power of ANS representations to reproduce expert behavior when they are properly designed and trained. To accomplish the same level of performance for the ACM Expert System, methodical generation and selection of the SAAC data will be necessary to ensure consistent pilot behavior and an even distribution over the solution space.

Training Parameter Optimization

There are two basic phases to optimization: initializing the weights and selecting the training parameter values. In both cases, the goal is to get the network to converge on the global minimum as quickly as possible. By trying different values for the initial weights, the chances are increased that a path to the global minimum will be found. The optimization of the initial weight ranges serve to accelerate the process by beginning the gradient descent from an optimal point on the weight surface. The optimization of the training parameters (the learning rates, momentum terms, etc) is also motivated by reaching the local minimum, and hopefully the global minimum, as quickly as possible. However, the effect of the learning rates and momentum terms on the update process can be quite complicated. Slow learning rates (low α values) ensure that the minimum well is not skipped by causing the descent process to evolve in a smooth, continuous fashion, but the number of required iterations can be greatly increased. A faster learning rate may jump over local minima and proceed more quickly to a solution, but may also skip right over the global minimum.

It may seem as though any values chosen for the initial weights and the network training parameters will be sufficient to produce an optimal solution given enough training time. In this view, the only effect of the variables is on the amount of time required to reach a certain level of performance. In some cases, this may be true because a complex weight structure may have many pathways to the global minimum. But since the gradient descent process does not always proceed monotonically across the weight space to the global minimum, the process may settle into local minima wells, or it may proceed very slowly over relatively flat areas of the weight space. This means that the choice of the initial weight values and the variables which control the movement across the weight space has a very pronounced outcome on the final performance of the network. It was also discovered that with the creation of ARCADE networks, it was essential that batching be selected as the weight

update process. While it is true that the gradient descent weight update process was designed with batching to be used, single iteration weight updates, nonbatching, has been a close enough approximation to solve many other problems.

Amount of Training

In general, the more time spent training the network, the lower the mean squared error and the better the final performance of the network. Since MSE reduction is an asymptotic process, as was shown in Figure 6, the amount of improvement in the mapping will also flatten out over time. In the experiments reported previously, training was carried out until MSE seemed to have flattened out. The number of iterations required to reach this level is highly dependent on the current problem and the training set being used. Time requirements are based on both the number of iterations and the number of interconnections that must be updated in the network. For the types and sizes of problems faced by the early ACM Expert System experiments, it generally required between ten and thirty thousand iterations to converge on a solution to the mapping. Time requirements were on the order of a few seconds. Solutions to the Lead Pursuit/Intercept problem required hundreds of thousands of iterations, which translated to ten to fifteen minutes of training. Solutions to ARCADE required millions of iterations, which translated to a few hours of training.

For complex problems, the minimization of network training MSE and MAE is not be the ultimate goal in network training. For those problems where the training data is only a sample of the operating domain, the optimal network will be a set of weights prior to the minimization of network training MSE. A network which has the lowest test MSE/MAE will perform better over-all.

Summary of Network Performance

At this point, it seems very likely that neural networks can provide a unique and workable solution to some important elements of the ACM Expert System problem. The results documented in this report show that ANS architectures can successfully capture a wide range of expertise and form generalized responses to input conditions. ANS architectures provide a way to overcome the knowledge engineering bottleneck, they produce robust solutions even under novel circumstances, and they are relatively quick and easy to implement. It is evident, however, that one neural network alone is not sufficient to produce a diverse and reliable model of pilot expertise under the full range of air combat conditions. It is expected that some hybrid of artificial neural systems and rule-based expert systems will eventually be used to provide a much higher level of cognitive simulation than either one could provide individually.

END